SICS Technical Report T2012:05 ISSN 1100-3154

SWEDISH INSTITUTE OF COMPUTER SCIENCE

Federated Robust Embedded Systems: Concepts and Challenges

Avenir Kobetski and Jakob Axelsson
{avenir, jax}@sics.se
Swedish Institute of Computer Science

2012-06-29

Abstract

The development within the area of embedded systems (ESs) is moving rapidly, not least due to falling costs of computation and communication equipment. It is believed that increased communication opportunities will lead to the future ESs no longer being parts of isolated products, but rather parts of larger communities or federations of ESs, within which information is exchanged for the benefit of all participants. This vision is asserted by a number of interrelated research topics, such as the internet of things, cyber-physical systems, systems of systems, and multi-agent systems. In this work, the focus is primarily on ESs, with their specific real-time and safety requirements.

While the vision of interconnected ESs is quite promising, it also brings great challenges to the development of future systems in an efficient, safe, and reliable way. In this work, a pre-study has been carried out in order to gain a better understanding about common concepts and challenges that naturally arise in federations of ESs. The work was organized around a series of workshops, with contributions from both academic participants and industrial partners with a strong experience in ES development.

During the workshops, a portfolio of possible ES federation scenarios was collected, and a number of application examples were discussed more thoroughly on different abstraction levels, starting from screening the nature of interactions on the federation level and proceeding down to the implementation details within each ES. These discussions led to a better understanding of what can be expected in the future federated ESs. In this report, the discussed applications are summarized, together with their characteristics, challenges, and necessary solution elements, providing a ground for the future research within the area of communicating ESs.

Workshop participants

Kristian Sandström (ABB), Kurt-Lennart Lundbäck (Arcticus Systems), Markus Wallmyr (Cross Control), Joel Huselius (Enea), Stefan Eck (Mimer), Dag Nyström (Mälardalen University), Jakob Axelsson (SICS), Avenir Kobetski (SICS), Kent Melin (Volvo Cars), Joakim Fröberg (Volvo Construction Equipment), and David Rylander (Volvo Technology).

Table of Contents

1 Introduction

This report summarizes the results from a pre-study project called *Federated Robust Embedded Systems with Quality and Efficiency (FRESQUE)*, aiming at providing a broad understanding of the challenges related to making embedded systems connected and their functionality adaptable.

1.1 Motivation and background

Embedded systems and software are becoming vital parts of many products, and have fundamental importance when it comes to improving their efficiency and effectiveness. Traditionally, embedded systems have been isolated parts in a single product, controlling that product through actuators and based on its own sensor data. However, with the arrival of cheap communication technology towards the surrounding world, both wired and wireless, it becomes possible for the embedded system of one product to exchange large amounts of data with other embedded systems. A substantial part of the embedded system's sensor data and internal state can now be exposed to the environment. The communication also makes it realistic to extend or update the embedded system's functionality after its initial design and deployment.

This development is illustrated well by the last year's explosion of so-called "apps" for "smart" mobile phones, i.e. small pieces of software that are developed independently and installed on a generic platform. The apps make it possible to continuously add new functionality and extend the user's perception of the world through data gathered on the Internet and from services placed on server computers in what is sometimes called the "cloud". These technology steps have released an enormous creativity and contributed to many innovations within mobile information systems. In fact, it has led to the creation of a whole new ecosystem of companies creating new services to extend the capabilities of the original device.

The same possibilities have, to some extent, started to find its ways into embedded systems as well, but we are most likely facing a large scale introduction within the coming years, where also traditional products will start to make use of increasing amounts of information and flexibility. Such embedded systems will at different points in time form "federations" with other systems, both embedded ones and traditional IT. To do this for business critical and safety critical systems has however totally different requirements on robustness and quality than to do it for consumer electronics. At the same time, it opens up new possibilities for product development organizations to gain access to data on how their products are actually used in the field, enabling optimizations and updates to strengthen the products' competitiveness.

1.2 Scope

The purpose of this project is to investigate what mechanisms must be put in place to make it possible for embedded systems to become federated and open for functionality extensions through software add-ons, without compromising with the special requirements that apply to such applications, such as real-time properties, limited resources, fault handling, etc. This includes technical solutions, primarily in software, which can be used by the systems, but just as important are things like architecture, development methods, and business aspects. The evolution described above makes the previously isolated products parts of a system-of-systems, leading to a dramatically increased complexity where far more cases must be handled, including unforeseen situations and (sometimes unintended) emergent behavior. A different way of thinking about robustness is necessary, together with new analysis methods to allow development to be carried out efficiently.

Outside the scope of this investigation lies the invention of new base technologies, e.g. for communication, but it is assumed that existing technology is sufficient. Focus is instead on the higher levels of system development and includes both solutions in the embedded systems and on servers outside the individual products.

1.3 Goals

The specific goals of the project were to:

- Identify a wide range of potential applications to motivate investment in this technology
- Identify key requirements of those applications that must be met by the technology
- Identify generic challenges that are cross-cutting the application domains

1.4 Approach

The pre-study project resulting in this report was organized as a series of five one-day workshops where different topics were discussed in order to identify important concepts and challenges. The workshops were organized by the Swedish Institute of Computer Science (SICS), and the additional participants were from ABB, Arcticus Systems, Cross Control, Enea, Mimer, Mälardalen University, Volvo Car Corporation, and Volvo Construction Equipment.

At each workshop, the participants brought in a number of concrete ideas of applications used to trigger insights into how a federation of embedded systems should be organized. The discussion then moved step by step down the ladder of abstraction, starting at the top level of a system-of-systems, and ending in implementation considerations within the embedded systems.

In parallel to the workshops, a state-of-the-art study on related research was carried out, including publications from many research domains such as cyber-physical systems, systems-of-systems, Internet of Things, ubiquitous systems, sensor networks, and agent-based systems [1].

1.5 Report overview

The report is structured as follows. Chapter 2 describes a number of application scenarios where mutually interdependent embedded systems are the main players. In Chapter 3, key characteristics of federated embedded systems are outlined, on the basis of the workshop discussions. This includes such topics as conceptual component model, life-cycle and ecosystem considerations, as well as reference architecture. In Chapter 4, main research challenges are collected, including both technical and non-technical questions. Finally conclusions are summarized in Chapter 5.

The report also contains one appendix, with a more detailed description of the most discussed application examples.

2 Application portfolio

In this chapter, a number of applications where interactions between embedded systems play an important role are shortly described. The list of application examples makes no attempts at being exhaustive, but is rather a selection of future visions within several technological fields, including energy, transportation, urban, healthcare, manufacturing, and agricultural domains. Most of these applications were discussed during the course of the workshop series.

2.1 Automotive applications

2.1.1 Route planning

The goal of this application is fairly straightforward. The idea is to aid drivers in choosing the optimal route according to their preferences (e.g. fastest, cheapest, etc.) to their destination points, taking into account the information about the surrounding traffic. The route planner could be either central, distributed to each vehicle, or something in between. This application has connections to most other automotive applications, either giving them an input or using their output information.

2.1.2 Smart traffic lights

The goal of this application is to reduce congestion and fuel consumption in traffic situations by controlling speeds of the vehicles as they approach intersections governed by smart traffic lights. The idea is that the approaching vehicles share some information, for example about their speeds, positions, and routes, with each other and with a traffic coordinator. Such a coordinator could be located in a traffic light, at a remote server, or in the vehicles themselves.

The coordinator will set a reference velocity for each vehicle's passage of the intersection, preferably forwarded internally to a cruise control mechanism. In other words, the coordinator is supposed to actively control the vehicle speeds. However, it is the responsibility of each vehicle to maintain basic safety. For the sake of flexibility, the traffic light periods can also be adjusted to the traffic flow variations. Also, it seems natural to connect the intersection planning that the smart lights offer with the route planning application.

2.1.3 Cooperative pre-crash systems

In this application, vehicles monitor the speed and actions of their own drivers, as well as speeds and behaviors of nearby vehicles. If the risk of collision is detected, the driver is first warned via visual, audio or haptic signals. Unless the driver manages to solve the situation in time, the vehicle tries to actively avoid collision.

If collision is unavoidable, the in-car safety systems should get activated in advance. Such actuators as air bags, seat belt pre-tensioners, and extendable bumpers should be used in an optimal way with respect to the imminent crash situation. This presupposes additional information exchange between the vehicles involved in the collision. The information of interest is for example a more precise position information, vehicle size, passenger placement, etc. Nearby rescue teams and hospitals would also benefit from appropriate information.

Both the US and EU authorities are planning to stimulate or even legislate the use of such cooperative pre-crash functionality.

2.1.4 Intersection collision avoidance

In this application, intersections are monitored by roadside equipment that collect and share the information about approaching vehicles. On receiving an alert signal about a vehicle approaching at a high speed, the vehicles on the intersecting road should slow down or stop in time. Taking this one step further, it should be possible to communicate directly vehicle-to-vehicle without the need of intermediary equipment.

An example of a research program that addresses issues related to this application is CICAS (Cooperative Intersection Collision Avoidance Systems) [2], involving a number of US universities and large automotive companies. The goal of CICAS is to develop technology that is able to warn drivers about likely violations of traffic controls, as well as to help in maneuvering through an intersection (without actually going so far as to taking control over vehicles). The technologies chosen by CICAS include both vehicle-to-vehicle and vehicle-to-infrastructure communication, using DSCR (dedicated short range communication) standards for message passing. HMI aspects, related to when and how to warn the driver about dangers, were also considered.

2.1.5 Cooperative driving / platooning

Platooning, i.e. driving vehicles in train-like formations, with the first vehicle in the train being the leader that the other follow, has been a hot topic for a while. Many automotive companies have been involved in developing prototypes and today the technology for making platooning a reality has matured quite a lot. For example, during the 2011 Grand Cooperative Driving Challenge (GCDC) [3], a competition between automotive research groups, platoon prototypes that communicate using IEEE 802.11p standard have been demonstrated. Today, much effort is put to maintain safety in a platoon, preferably through graceful degradation, in case of vehicle or communication equipment malfunctioning.

The platoons participating in GCDC are generally pre-designed, pre-tested, and consist of a fixed constellation of participant vehicles. The situation becomes much trickier if we consider real-life settings, where vehicles of different manufacturers and owners may enter and leave platoons at any time. Also, the number of participants in a platoon, as well as the number of nearby platoons will be much higher, with increasing challenges with respect to communication. Natural questions are how to establish and close a communication channel, how to handle interference problems, bad coverage, etc. Also, in a realistic setting, unprotected road-users must be carefully considered.

Once the technology for driving a platoon safely is in place, the next challenge is the question of how to form and dissolve platoons in an efficient way that gives each participating vehicle some kind of profit. It is imaginable that there will be some requirements for entering a platoon, for example regarding the vehicle size, intended speed, maximal acceleration, route, readiness to participate in platoon's business model, etc. Altogether, this makes the formation planning a complex scheduling problem that evolves dynamically according to the traffic situation, platoon requirements, and changing intentions of the vehicle drivers. Platoon scheduling seems interconnected with the route planning problem, with the links becoming more dominant the longer distance the vehicles plan to travel. For example, for a long-haulage truck it will often be worth to take a detour or delay its schedule in order to participate in a platoon that moves towards its destination. Also, smart traffic lights should take into account the existence of platoon formations, to avoid breaking up platoons due to bad intersection management.

An efficient procedure for the appointment of a leader vehicle will also be needed when a new platoon is formed, when the leader vehicle leaves its platoon, or when the leader moves backwards through the platoon to draw advantage of its wind shield. Of course, the question of fair profit distribution, and thus the choice of a leader, can be solved using some sort of business model. In some cases, it might be optimal to maintain one leader vehicle throughout the whole journey of the platoon, for example if that vehicle provides the best wind coverage for the followers or if it has the best road detection systems, allowing the whole platoon to move more smoothly. In such a case, it is imaginable that the follower vehicles pay for their part in the platoon. This could probably be done in different ways and is an interesting question of its own.

2.1.6 Emergency lane clearing

This application uses the new communication technology to address a familiar need. The idea is to transmit a warning message to the vehicles that are travelling along the route of an emergency vehicle about its imminent approach. Ideally, this should be used in parallel with route planning. First of all, the emergency vehicle itself may change its route if the traffic situation ahead dictates so. Second, the vehicles that are approaching the planned emergency lane may be redirected.

2.1.7 Hazardous road conditions notification

Information that may affect the traffic flow should be forwarded to all concerned vehicles. For example, this could include information about traffic situations, road conditions (slippery, holes in the pavement), obstacles on the road, near-road animals, weather information, road slopes, road-side construction work, etc.

It might be useful to aggregate the road condition information centrally to get a more stable picture of the environment. The central data processing unit could also be allowed to trigger some pre-defined actions in response to certain data patterns. For example, requests for snowplows may be placed if a sufficient number of vehicles are signaling about difficult road conditions due to snow. This could also be done pro-actively by reacting to weather forecasts.

The spreading of information about road-side construction could be facilitated by the construction machines acting as servers. Also, if the construction work leads to a traffic jam, this should be detected, and communicated to the approaching vehicles. This is achievable even with a small number of vehicles equipped with some sort of sender technology.

Traffic information can and should be used for route planning, either in a centralized or decentralized fashion. This can be further extended to consider the transportation system in its entirety.

2.2 Home and energy automation

2.2.1 Smart appliances

A smart home, based on the vision of ubiquitous computing, contains a number of smart appliances that can interact with the user, with each other, and with the outside world to improve the quality of life for the smart home inhabitants. The key lies in the processing of information, which can be collected from the other appliances or from the outside world, in an intelligent way. Ideally, the smart appliances will learn and adapt to the individual preferences of the inhabitants, serving the user in a more appropriate way and with less disturbances.

An example of a smart appliance is a refrigerator that proposes recipes for each inhabitant using the groceries that it contains and taking into account such factors as age, gender, and Body Mass Index [4]. If a recipe is acknowledged by the user, a smart oven is automatically configured with the appropriate settings. Such refrigerators exist already. However, as of today, they only communicate with the appliances that are produced by the same company. In the near future, it is believed that appliances of different producers will communicate with each other.

Another example is the adjustable lighting that turns on sporadically when the inhabitants are away, in such a way simulating their presence and discouraging potential thieves. Heating costs could be reduced by automatically lowering the indoors temperature when there is nobody home. The heating could then be brought back to a normal level when one of the inhabitants is approaching, which could be communicated automatically by the user's smart phone.

2.2.2 Smart grids

The key idea of a smart grid is to efficiently balance production and consumption of energy, avoiding electricity shortages, but also avoiding over-sized energy production facilities that are needed today to handle demand peaks. When discussing smart grids, it is often assumed that the energy production is distributed, with a large number of small producers, often at the level of an individual (smart) home, which makes the scale of the federation an important challenge.

It is imaginable that the producers, even the small ones, have some sort of energy storage capacity. Small-sized producers are typically free to choose when to sell their surplus. Naturally, they will try to maximize their profits by getting as good price as possible when selling the electricity, while fulfilling their own consumption needs. Typically, this will include keeping track of the electricity price forecasts, predicting future electricity production, scheduling the electricity consumers to run when the price is low (whenever possible), and even joining so-called virtual power plants, i.e. federations of small producers that help each other to attain a larger profit. Grid operators, on their hand, will use pricing to counter peaks in both the electricity supply and demand. They could also actively request lower energy consumption or even shut down some non-critical network sectors in case of system failures to prevent blacking out critical services, such as hospitals. The scheduling problem is non-trivial due to its scale and inherent instability, caused by conflicting and rapidly changing conditions.

A typical example of how energy consumption can be balanced is to schedule the washing machines to be run at night, when the electricity price is relatively low (however at the same time the CO2-footprint is higher due to a larger portion of coal energy). Also the dishwasher activity may be postponed if profitable, even if the scheduling window is generally narrower in this case than for the washing machine. However, while some electricity consumers could be run in a flexible way, subject to their individual constraints (both technical specifications and user preferences), other consumers, such as lighting, will not be schedulable. An interesting example of home appliance scheduling, subject to realistic technical specifications, can be found in [5], where a dryer, a washing machine, and a dishwasher are scheduled in an optimal way with respect to a 24-hour electricity price forecast.

Another component of a smart grid, that is expected to gain in importance, is the fleet of electric vehicles (EVs). In fact, EVs are both a plague and a beneficiary to the smart grid. To be practically useful, EVs must be able to charge rapidly. This leads to high loads on the electricity net, with take-

outs reaching 32 kWh in a few hours, which is comparable with the daily consumption of a typical house that lies around 20-50 kWh [6]. Also, EV-induced loads are prone to aggregate, both in time and geographically. For example, EV charging loads are assumed to peak in the afternoons, when people return home and plug in their EVs. Geographical load concentrations occur in cases of social events when a large number of EV drivers gather and simultaneously plug in their vehicles.

On the positive side, EVs offer a more powerful tool than ordinary home appliances to counter energy peaks by selling the energy back to the net when necessary. Of course, this should be done without compromising the user's travelling needs. An interesting review of the challenges that smart grids pose can be found in [7], where EV pros and cons, virtual power plants, demand management, and self-healing networks are discussed from the AI perspective.

2.3 Healthcare

The number of applications of information and communication technologies for better healthcare is growing rapidly, with different buzzwords being circulated, see e.g. [8]. Some examples are *eHealth*, *tele-medicine*, *mHealth*, *health knowledge management*, etc. Generally, eHealth includes all the other terms, telemedicine is the term that fits our setting best, while mHealth could be defined as telemedicine that is enabled by the use of mobile phones.

2.3.1 Telemedicine

Telemedicine consists of three mutually complementary categories:

- *Store-and-forward telemedicine* consists of acquiring medical data and transmitting it to a medical specialist for an (offline) analysis. A medical record should preferably be attached or retrieved from a health database. If this could be combined with the use of smart phones, thus making the technology widely available, this could be the new way of making the first visit to the doctor. In some cases, the role of a specialist could be played by a computer agent.
- *Remote monitoring* stands for a continuous monitoring of health conditions in home environment. Normally, it is thought to monitor patients with chronic diseases (e.g. pacemaker users, diabetes patients, etc.) or in elderly care. An example of remote monitoring in elderly care that makes use of smart home sensors is to detect and react on abnormal movement patterns of an elderly person within its apartment.
- *Interactive telemedicine* represents real-time interactions between the patient or an elderly person and the care provider. An example application is the Giraff-robot [9], developed with the support of Robotdalen [10], which is a mobile Skype-enabled screen that simplifies the remote contact between elderly people and their friends, family, and the home-help service. Another example is the possibility to offer specialist care, including surgery [11], remotely.

2.3.2 Smart pills

An application that could be sorted under remote monitoring is spelled *smart pills*. The idea is to let a pill act as a transmitter that notifies a monitoring system when reaching the patients stomach. If such notification has not arrived in time, an alert signal is sent back to the patient, reminding about the time for medication. The technology has already been demonstrated using biodegradable ingredients that send out a digital signal on reaction with stomach fluids [12]. Another approach uses small amounts of silver, no more than normally can be found in a glass of water [13].

Smart pills have also been developed with the purpose of making gastric diagnoses in a store-and-forward manner [14]. A smart pill is then capable of measuring pH, temperature, pressure, etc., while passing through the digestive system, and reporting the results to a physician.

The next logical step is to advance the intelligence in smart medicaments so that they automatically adapt to the patients conditions. An obvious example would be to aid diabetes patients to regulate their blood sugar balance.

2.3.3 Medical device "plug-and-play" interoperability

In a modern hospital, a large number of heterogeneous devices are used to aid in the diagnosis and treatment of patients. Today, this is often done in a stand-alone fashion, where each device is produced and configured on its own. In the future, medical devices will share their information with each other to improve the quality and efficiency of care.

Already, the idea of third-party applications that will add plug-and-play interoperability to medical devices is being discussed [15]. As it looks today, the main challenges lie within performance and safety in large medical systems of systems, as well as in standardization, with quite some efforts done towards a medical interoperability standard ISO/IEEE 11073 [16].

To give a more specific example, in a futuristic scenario, a nearby hospital will be automatically notified about the place and severity of a traffic accident. Appropriate equipment and medical professionals can then be automatically configured in time for the arrival of the injured to the hospital. This may include preparation for remote/assisted surgery. Any allergic reactions and medical records of the injured will be automatically analyzed to prevent inconsistent drug combinations.

2.4 Transportation

2.4.1 Demand driven multi-modal public transportation

Today, public transportation information normally moves in only one direction, i.e. travelers are (in the best case) notified about real-time traffic information. The aim of this application is to close the loop and let the varying traveler demands, plans, and traffic situations to actively affect the public transportation schedules. Connections should be coordinated, preferably in real-time, so as to improve the overall transportation efficiency.

The coordination should not only be done between the transportation entities of the same type, but also extend beyond the transportation modes, transport operators, and even regional and state limits. Continuous (online) coordination will be needed in some situations (for example, a bus may be requested to stay longer at a stop waiting for a delayed connection), while more long-term transport planning will be assisted by detecting patterns in the recorded travelling information.

A move towards traveler driven transportation came with Flexlinjen in Göteborg, a bus service originally directed towards elderly people. It allows calling in and requesting a bus to a Flexlinjen stop. In such a way, buses are only run if needed, while elderly people can reserve a seat before the arrival of the bus. While important in many aspects, the Flexlinjen initiative is still quite manually run and serves only a small portion of public transportation needs.

2.4.2 Taxi sharing

In several major US cities, taxi sharing smartphone apps are gaining popularity [17] [18]. When such an app is launched, a list of people that match the user's origin and destination points appear, enabling to agree on a meeting place for taxi sharing. Sometimes, there is an option to rate travelling companions for future needs. As of today, the booking of taxi and the choice of appropriate companions (and thus also route planning) is not automated.

2.4.3 Private car sharing

This application is closely related to the taxi sharing, with the idea of bringing drivers and passengers together. It works in such a way that private drivers announce their planned routes and times of departure, while potential passengers search through such routes and contact those drivers that match their planned itinerary. If there is a match, the passengers pay some fee to the driver, preferably using the same app.

A prototype of a car sharing app, Citihaiq [19], was launched in 2010 by a professor in chemistry. However, it never took off, due to both bad performance (the app often hanged the smartphones) and an unpopular business model (15% share of each passenger fee would go to the app developer).

2.4.4 Personal Rapid Transit systems

To better exploit the benefits of a demand driven transport coordination, there should be a high flexibility in the public transportation system. This includes not only information passing between different transportation providers but also diversification of available transportation modes. A move in this direction is the use of smaller and more flexible transportation entities that can be assembled together into larger trains.

Personal Rapid Transit (PRT) systems [20], deployed among others in London Heathrow airport and in Masdar city, are interesting examples of this idea. PRTs are rather small, train-like vehicles that run on rails. Typically, they are built to carry 3-6 passengers, even though larger wagons exist that can take up to 30 passengers. Normally, PRTs are reserved exclusively for the travel of an individual or a group of individuals, moving directly from their origin to the destination. During the transit, it is quite common that several PRT wagons that are moving in the same direction are clustered into platoons. In fact, the similarity to the problem of platoon formations between privately owned vehicles seems quite large, even though the coordination is probably significantly simpler in the PRT case due to a less intersections and destination points.

2.5 Manufacturing and construction

2.5.1 Automation in the Cloud

The idea is to outsource hardware management from traditional manufacturing environments to central provider of cloud services. This would lead to cost reductions for the manufacturing companies, both when it comes to the hardware costs, as well as to the maintenance and configuration costs. It would also open possibilities for coordination of production lines between the facilities. For example, if a factory in Sweden and in Australia are working in a similar way and share their information, the time difference could be used to run production continuously, with lower capacity requirements as a result.

Unresolved questions with this kind of approach are among others security, information ownership, and business models. Real-time requirements in this kind of distributed environment may also pose significant challenges.

2.5.2 Flexible manufacturing

Flexible manufacturing is a trend in the modern production systems. Intelligent planning of the manufacturing processes allows working on different products in the same production line. Just-in-time production and logistics planning are well-known concepts. However, this could be driven even further by interconnecting all sources of information that may improve the manufacturing process, including manufacturing equipment, delivery vehicles, order systems, stock trading systems, etc.

In such an interconnected system, the order and delivery of spare parts will be automatically coupled to a production order being placed. Furthermore, predictions of future demands, or similarly predictions of future costs of raw material and spare parts will also affect the production planning. The delivery to the factories will be traced in real-time. Equipment failures will lead to automatic reconfiguration of equipment or production re-planning, including moving some parts of production to other facilities (e.g. using the benefits of the automation in the Cloud concept). The information about equipment failures will be gathered and analyzed.

2.5.3 Construction equipment systems of systems

At a modern construction site, equipment of different types and brands is often used to accomplish a common task. This leads to the desire of being able to easily interconnect different machines, in a kind of plug-and-play fashion. This would open up for more flexible construction sites, with simple introduction of new or rented machines if necessary.

Taking a step beyond the plug-and-play interoperability, construction machines should be combined together in a way that optimizes some operational performance criteria, such as efficiency, cost optimality, etc. In fact, today it is increasingly important to take a holistic view of the whole construction site since the technology has reached a point when it is difficult to make major improvements with individual machines. Such optimization requires both smart algorithms and machine performance data becoming public.

Equipment manufacturers on their hand are inclined to be reluctant to let their products share too much information with the machines of competitor brands. While performance data could have been used for the optimization of machine-to-machine interplay, it risks revealing certain technological secrets. Thus, information handling and ownership, as well as security issues are important questions.

2.5.4 Warehouse automation

An example of warehouse automation was briefly discussed during the workshops. In this application, laser guided vehicles (LGVs) transfer bottles and cans within the Carlsberg warehouse in Falkenberg. LGVs find their position using laser reflectors, while their tasks are planned by a central computer, in a similar way to a taxi booking system. LGVs are run on batteries that are automatically scheduled for charging or replacement.

This application resembles the above discussed construction equipment systems of systems. However, it only contains machines of one manufacturer, which removes the data sharing and security challenges.

2.6 Forestry and agriculture

2.6.1 Precision agriculture

The key idea of precision agriculture is to take the variability of agricultural parameters into account. This is especially pronounced in large farms, which probably explains why the first steps within this domain came in US, Canada, and Australia. Typical parameters, which may vary in both time and space, include weather conditions (drought, rain, etc.), soil characteristics (texture, depth, nitrogen levels, etc.), weed incidence, disease outbreaks, crop ripeness, etc.

Using some sort of variable-rate farming equipment in an intelligent way, growing conditions can be optimized with a high precision. For example, the use of fertilizers can be precisely managed to refill only those farm zones where the nitrogen levels are low, with both economic and environmental benefits. If plant diseases are detected in time, they might be countered with a small or even zero doses of pesticides. Weed control can be done easier and more ecologically. An interesting example of this is a weed burning robot that can distinguish between certain types of plants and weeds, literally burning the weeds while sparing the plants [21].

An envisioned precision agriculture farm will be dependent on a number of interacting components, such as:

- wireless sensor networks;
- small autonomous robots with sensing and communication equipment;
- larger machines conducted by human drivers (these machines may even lack the communication equipment);
- positioning systems, e.g. GPS;
- geographic information system (GIS) that analyses the sensory information about the state of the farm and make decisions about proper future actions;

It is also imaginable that parts of this equipment is rented for short periods of time or shared between several farm owners. Further, the GIS may interact with a larger information system, which could include other neighboring farms, Cloud data, or even internet. A simple example is the extraction of long-term weather forecasts from the internet to improve the decision making.

2.6.2 Autonomous agricultural vehicles

Applications of fully autonomous vehicles or vehicles that are aided in their movement by a computer system are finding their way into the agricultural domain. Such vehicles are often a part of the precision agriculture vision, but they are also interesting in their own right.

One example is a European research project, aimed at improving algorithms for the so-called ambient awareness and obstacle detection, making use of the sensory information (vision, radar, thermal, etc.) [22]. Another is an auto-steering system for cultivators, which makes sure that the cultivator follows a straight row even if the tractor does not [21].

2.6.3 Tractor-to-equipment synchronization

In many agricultural tasks, a tractor is closely cooperating with other agricultural equipment, such as a cultivator, plow, combine, etc. Ideally, some information from the hanger-on equipment could be displayed to the tractor driver in real time. The challenge here is to make sure that the additional information does not disturb safety critical parts of the tractor's operation.

Another interesting application is the synchronization of movements between different agricultural machines. An example is to let a combine control a grain cart (pulled by a tractor) in a master-and-slave way, i.e. the tractor follows autonomously the movements of the combine while the crop is harvested and loaded onto the grain cart. An application providing this functionality and also allowing up to 10 machines being combined into a data distribution network was recently released to market [23]. The data sharing functionality gives the famer a better overview over the state of its machinery and leads to better (yet still manual) decisions.

2.6.4 Timber harvesting

Machine-to-machine communication finds a natural application in the forestry domain since the timber harvesting process consists of a number of steps (felling, extraction, processing, loading, and trucking) that each involves different machinery. In a way, this application is related to the construction equipment systems of systems (see Section 2.5.3), since the equipment of different manufacturers need to interact, preferably in a plug-and-play manner.

However, the geographical field of operation is generally larger in this application. For example, it is desirable that the felling machinery sends the GPS position of every fallen tree to the extraction scooter, which can be quite far away at the time of signal transmission. Also, the radio coverage tends to be weaker in a forest area. Thus, communication challenges grow in importance. This includes development of standardized communication protocols that account for transmission losses, balancing between local and global communication technologies, etc.

2.7 Other applications

2.7.1 Cyclist protection

This application is intended to warn drivers of turning vehicles about cyclists that are located in their blind spots. Even though there exist video and radar based solutions to this source of accidents, this kind of application could be used with older vehicle models, through the use of a smartphone.

The idea is that the cyclist's smartphone emits a position signal that is received and interpreted in the turning vehicle. One challenge lies in the communication, and specifically in how to handle transmission losses safely. Another challenge is related to the interpretation of the position signal. The risk of false alerts seems high in a crowded city, which would make the application useless and could even be dangerous by distracting the driver unnecessarily.

2.7.2 Autonomous public service vehicles

Automated guided vehicles (AGVs) have been used for quite some time in manufacturing applications. Today, their role is starting to reach out into other domains, such as agriculture, forestry, mining, etc.

Here, the focus is on vehicles providing some sort of public service, with typical examples being autonomous snowplows or autonomous lawn mowers. Several proofs of concept exist, not least thanks to the yearly student competitions in snowplowing [24] and lawn mowing [25], with the vehicles being automatically guided either using road side markers or GPS. Although the idea is quite simple, the implementations pose some technical challenges, with the safety considerations possibly being the most important challenge. Also, GPS devices may fail or show inaccurate information if the satellite signal is blocked in some way, for example by tall buildings, trees, or atmospheric conditions.

The autonomous snowplow idea could be extended to platoons of autonomous vehicles. For example, some airports have snowplowing forces in a standby mode during winters, in order to rapidly clear landing tracks. Such snowplows generally move one after another with only some centimeters of lateral displacement. Apparently, a better synchronization with the lead vehicle should be possible, making the procedure much more efficient.

2.7.3 Grocery tracking

The idea of this application is to present all relevant information about groceries to the customer in a blip of a scanner. This could include price, place of origin, total CO2-emissions during transportation, additives, nutrients, allergic substances, corporate policies, etc. The customer could set some preferences before going to the store, allowing the system to warn about the groceries that do not fulfill the customer's desired profile.

The technological prerequisites to achieve this kind of scenario exist. The main challenges seem to be related to the information itself. Firstly, the amount of information that needs to be handled would be huge. Secondly, privacy issues regarding what the customers buy, as well as questions about targeted marketing, are important. Thirdly, it might be difficult to get the information from the whole production and transportation chain, at least in the beginning, due to corporate reluctance.

2.7.4 Cosm.com

Cosm [26], previously known as Pachube, is an online database that allows people (generally private hobbyists) to share sensory data and build applications using that data. In a sense, this is a platform that is perfectly in line with the FRESQUE vision of interconnected ESs. The Internet of Things is explicitly stated as the vision of Cosm.

The name change to Cosm stems from Pachube being purchased by a company named LogMeIn that develops technology for remote access to personal computers. Thus, it seems believable that Cosm may lead to even closer interactions between data sources and their users.

2.7.5 Military flight systems

Military flight systems offer an example application with high criticality, both physically during the flight but also when it comes to sensitive data that should not be shared. The last issue becomes especially accentuated in training operations with external military forces, during which systems of systems are often formed ad-hoc. The questions of what data could be shared, with whom, and when, are non-trivial.

2.8 Summary of common application characteristics

In this chapter, a number of application examples of federated embedded systems have been described. Apart from showing that the ideas have a potential across a wide range of applications, the examples are aimed as a portfolio that will prepare the ground for future research on how to efficiently and safely develop federated robust embedded systems.

Several of these applications were used as discussion facilitators during the workshops, leading to a deeper understanding and early conclusions about the recurrent characteristics and challenges within the emergent field of interacting embedded systems. Some of these characteristics have been described above, and in summary, the more important ones are the following:

- Robustness and safety
- Interoperability
- Communication channels, including unreliable channels
- Creation and dissolution of federations
- Business models and ecosystems
- Continuous control functions, but also use of aggregated data for planning
- Security, privacy, and ownership of information
- Centralized vs. decentralized control, and implications on timing
- Conflicting requirements between different embedded systems
- Large data sets

These characteristics reoccur in many of the examples, although not necessarily all of them in each and every application.

In the following chapter, common characteristics that were recognized during the workshops are condensed into a number of key concepts. In chapter 4, a number of research challenges are discussed on the basis of these characteristics.

3 Key concepts of federations of embedded systems

Based on the examples of federated embedded systems described in the Chapter 2, a number of key concepts have been identified. In this chapter, a basic terminology for modeling federations, individual federation components, and federation related stakeholders, is presented. This is followed up by discussions about life-cycle phases, development processes, and organizational structures. Finally, this chapter is concluded with a reference architecture description of common functionality in federations of embedded systems.

3.1 Conceptual model

In this section, key components of a federation of CS are presented using standard UML notation, separated into four groups by two conceptual axes, *Technology* vs. *Product & Process*, and *System-level* vs. *System-of-systems-level* concepts, Figure 1.

3.1.1 System-level technology

The starting point of this work, when it comes to the technology used, is an *embedded system (ES)* that interacts with other ESs, forming *federations* to gain some advantage that it cannot achieve individually. An ES is a *computer system (CS)* that is part of another system of some kind, such as a vehicle, an aircraft, some kind of machinery, or a power system. In other words, an ES is always embedded in some kind of *environment*, which consists of the surrounding hardware and the sensory information that the ES receives. Another kind of CS that does not interact directly with a physical environment but may play an important role in a federation, for example as a central server, is a *general-purpose computer system (GPCS)*.

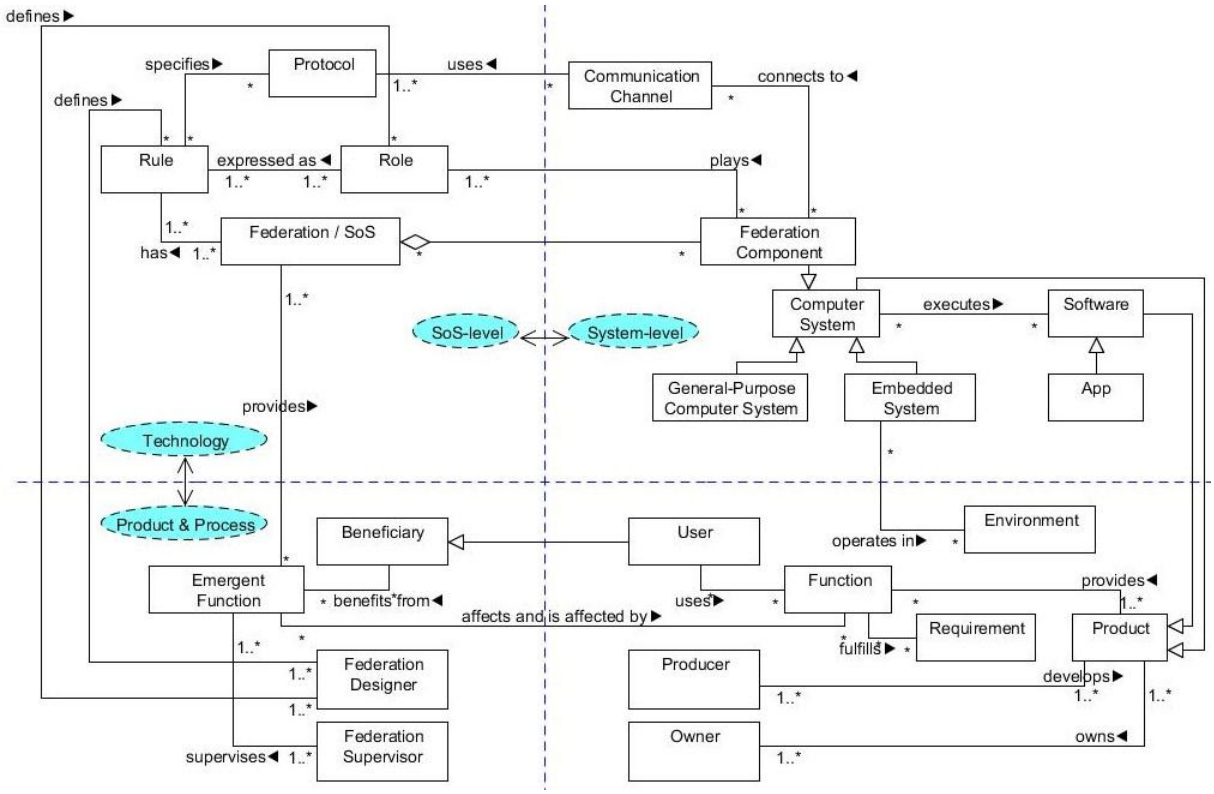


Figure 1. Conceptual UML-model of a federation of computer systems.

A fundamental pre-requisite for the formation of federations is the existence of some kind of *communication channel*. Furthermore, CS will generally need to execute some federation enabling *software* in order to interact in a meaningful way. In the setting, considered in this work, software functionality is supposed to be developed partly by the original CS manufacturer, and partly by a third party, possibly after the original CS deployment. Such additional third party software will be called *apps* in analogy with smartphone applications. Generally, app software will be tailor made for specific federation types. A CS that has both communication capabilities and software that is needed to form a federation will be called a *federation component (FC)*.

3.1.2 System-level product & process

The technological view is, on the system level, tightly connected to the product & process view. Each CS is a kind of a *product* that has been developed by a *producer*, is owned by an *owner*, and provides a number of *functions* to its *users*. The functionality of each individual product is regulated by either explicit or implicit *requirements*. To emphasize that software may be developed by another producer than the CS on which it runs, the software is considered as a product in its own right.

3.1.3 System-of-systems-level technology

On the system-of-systems level, the technology becomes more abstract and amounts to a careful definition of *roles* for the interacting FCs. Each role can be expressed as a collection of *rules*, generally implemented in an app, that specify the actual *protocols* used in communication between the FCs.

3.1.4 System-of-systems-level product & process

The notion of products is less applicable to the system-of-systems level since federations will not be produced in the word's classical meaning. Rather, they will evolve continuously, through a dynamic interaction of individual FCs, which are free to decide about when to join or leave a federation, as well as about how much they wish to contribute. Of course, in order to join a federation, FCs will need to comply with certain rules, defined by a *federation designer*.

These rules, together with the characteristics and functionality of the individual FCs will form the basis for the *emergent functions* of the federation. Normally, there will be one or several *beneficiaries* that draw advantages of each emergent function. Note that such beneficiaries need not necessarily be the actual FC users.

Some emergent behavior will be desirable and possible to anticipate, while some will not. Thus, there will be a need for *federation supervisors*, both in order to avoid compatibility problems before app deployment (offline supervision), and to be able to respond to hardware and software malfunctioning, as well as conflicts within the federation, during runtime.

3.1.5 Alphabetical concept list

Below, the conceptual elements of a federation, shortly discussed above, are presented in alphabetic order and defined more strictly.

App: Software that is added to a computer system after its initial deployment and contributes functionality that was not present initially. An app can be developed by another producer than the original manufacturer of a computer system. Although not required by this definition, in our setting an app will generally be used as a functionality enabler with respect to some federation.

Beneficiary: A person or an organization that benefits from the emergent functionality of a federation.

Communication channel: A transmission medium, either wired or wireless, for information passing between federation components.

Computer system (CS): A programmable machine, designed to automatically carry out a sequence of arithmetic or logical operations.

Embedded system (ES): A computer system, designed for specific control functions within a larger system. Today, many consumer products contain embedded systems. Cost considerations generally lead to restrictions in terms of available memory, processing power, energy consumption, etc. An ES is said to be embedded in an environment.

Emergent function: High level functionality of a federation, shaped by federation designers through the definition of rules and roles for federation components.

Environment: Physical surroundings of an ES that in some way can affect that ES or be affected by it. Generally, a product containing an ES will be considered to be a part of the ES environment, while mechanical parts, such as sensors and actuators will be seen as the interface between the ES and its environment.

Federation / system of systems (SoS): A collection of federation components that follow certain rules and play certain roles, as a result leading to the emergence of high-level functionality that would not be achievable by individual systems.

Federation component (FC): A computer system capable of participating in a federation through interaction with other computer systems**.** This includes both being equipped with the necessary hardware for communication and being able to install necessary software for the communication to be meaningful. Note that such definition does not necessarily require that an FC is always a part of a federation. It only states that an FC has mechanisms in place that allows it to become a member of a federation, provided that appropriate software is installed and that the FC chooses (or is forced) to join. Also, an FC is assumed to be self-sufficient, i.e. it has a reason for being and a functionality of its own outside of a federation. From this reasoning, it is natural to describe the status of an FC with respect to some federations as a collection of parallel state machines, see Figure 2. Each state machine represents one federation and to be able to participate, an FC will generally need to be extended with appropriate app. Once this is done, the FC is said to be enabled and can choose to actually participate in the federation by joining it. Of course, these steps should be reversible, i.e. an FC could leave a federation and even uninstall the enabling app.

Federation designer: A person or an organization that designs the emergent functionality of a federation by defining rules and roles of federation components. Often, the federation designer will also be the app producer, but it could also confine itself to producing a specification which is implemented as apps by other developers.

Federation supervisor: A person or an organization that has the responsibility for the correct functioning of a federation. If functionality corrections are needed, either at runtime or before app deployment, it is the federation supervisor’s responsibility to take appropriate measures, whether it

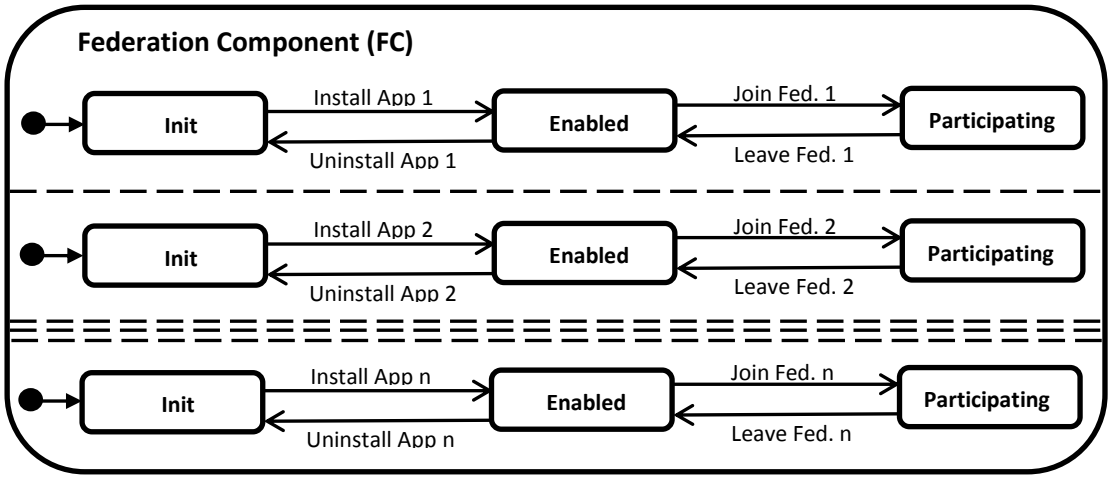


Figure 2. Inner states of a federation component.

involves contacting a federation designer, a producer, an owner, or some other part. The federation supervisor may also be responsible for the certification of apps related to the federation, something that will be referred to as offline supervision.

Function: System-level functionality of a product.

General-purpose computer system (GPCS): A flexible computer system, designed to meet a wide range of user needs. Examples include personal computers, servers, cloud technology, etc.

Owner: A person or an organization that has the legal ownership of a product.

Producer: Develops, and if necessary upgrades, a product. One product or function can be developed by several producers.

Product: The end result of a producer's development process. In our setting, interesting types of products are embedded systems, general-purpose computer systems, and software.

Protocol: A set of rule-based specifications (vocabulary + grammar) for the interactions between federation components over a communication channel.

Requirement: Mandatory functionality of the individual computer systems that may not be compromised by the rules of a federation.

Role: Federation components may have different roles in a federation. Roles are expressed as a set of rules, thus describing the obligations of associated federation components within a federation. Roles also define the organizational structure of a federation. A default role, treating federation components as equals without imposing any restrictions on them, is reserved for the case of no rules being defined (either explicitly or implicitly).

Rule: A federation has normally a number of rules that some or all federation components need to comply with in order to participate in the federation. In practice, rules will often be defined implicitly by the application software.

Software: Organized collection of data and instructions that is executed on a computer system and control its functioning.

User: Someone or something that operates a product or a function, whether it is a system function or a higher level system-of-systems (emergent) function.

3.1.6 Conceptual model of the smart traffic lights application

The smart traffic lights application, shortly described in Section 2.1.2, will be used here to illustrate the proposed conceptual model. Recall that the desired (emergent) functionality of this application is to control the vehicle speed when approaching a road intersection that is equipped with a smart traffic light, so as to reduce the number of stops and subsequent accelerations. An implicit requirement is that this should be done in a safe way.

There are at least two types of products involved in such an application, vehicles and traffic lights (both being embedded systems). The traffic light environment consists of the traffic light hardware, as well as the intersection that the traffic light supports, including the necessary information about nearby roads and vehicles. The vehicle environment includes the vehicles themselves, as well as nearby roads and vehicles.

The responsibility for controlling the vehicle speed could reside either in the vehicles themselves, in the traffic lights, or in a central (possibly cloud based) planner. To be concrete, we will assume here that the traffic control algorithm is centralized and managed by a cloud service provider, which is a third product type, belonging to the GPCS class. Further, it is assumed that the smart traffic lights application is governed by complimentary apps, downloaded to the vehicles, traffic lights and the planner after their initial deployment.

When it comes to the smart traffic lights apps, we will here assume that it is produced by a third party software development company, commissioned by the road administration authority (RAA). Since the RAA is probably quite interested in the emergent functionality of the federation, it feels natural that it also assumes the role of the federation designer and explicitly defines the rules for each type (role) of federation components before the actual app development. Similarly, since the service should be applicable to different vehicle brands, the testing and certification of the app should also be done by independent organizations.

Vehicle drivers are typical users of both the vehicles and the traffic lights, while potential beneficiaries of the emergent function are the drivers, carrier companies, persons living close to large roads, as well as the society as a whole.

At the individual system level, product ownership is often quite simple to establish. In our example, the user of a vehicle will in many cases also be its owner. At the federation level however, things become trickier. Ideally, there should be one or several federation supervisors that together are capable of monitoring the correctness and safety of all emergent functionalities. In our example, RAA is readily assigned this responsibility.

Now it is the RAA's task to monitor the state of the federation and initiate appropriate actions when necessary. For example, the information about malfunctioning traffic light hardware is probably best forwarded to the producer of the traffic lights (or pre-contracted service providers), while software bugs in the smart traffic lights app could be delegated to the app developer. However, simple as it seems to have one supervision authority, some borderline cases may still be somewhat unclear. For example, what happens if the app is malfunctioning only in combination with some vehicle brands? Should the responsibility be delegated to the app developer, vehicle manufacturer, or both?

When it comes to the communication technology, we assume that WLAN is used as far as possible, for example in communicating between vehicles and traffic lights. Since the planner is generally located outside of the WLAN coverage, 3G is used to communicate with the planner. To avoid charging drivers excessively for 3G communication, we assume that vehicles as far as possible communicate directly with the traffic lights that in their turn transmit necessary information about the vehicles to the planner.

From the product list it is natural to define the federation roles to be “vehicle”, “traffic light”, and “planner”. Each role is described by a collection of rules that will be further elaborated in Section 3.5.5, together with a reference architecture model.

3.2 System life-cycle

As for every ES, the design of federated embedded systems involves important considerations with respect to not just operation, but to all system life-cycle phases. In this section, these considerations will be detailed, with particular emphasis on those that differ compared to traditional ES.

Generally, the life-cycle of a product starts with some sort of development, goes on to a period of operation, interrupted at some points by maintenance, and ends in some sort of disposal. This can be mapped to the concept of federations, see Figure 3. In this work, we chose to focus on the aspects that are needed for the operation of a federation, leaving out disposal considerations. Life-cycle steps of a federation, as we see them, are presented in more detail in Figure 4 and described shortly in the following. Note that this life-cycle model includes normal ES life-cycle steps, for example normal ES (or CS) development, fault tracing, security handling, and repair.

3.2.1 Development

ES development: In the development of the base ES, all the normal activities need to be carried out to create the base application. In addition, it must be decided and implemented what data from the base application should be available to apps (e.g. sensor and state data), and what commands the base application should accept from apps (e.g. to control actuators or change state). Further, the execution environment of the apps must be included, and this will in most cases be a standard software module, but it must be connected to the data and command interface (CS-App API) of the base application. That interface must also be documented in some way, to present the capabilities of the device to app developers. Finally, testing must include possible ways that an app can affect the overall ES’s behavior to ensure sufficient robustness against erroneous apps. Reliability and trust mechanisms should be built in from the start, as well as safe modes to fall back on if an app behavior is considered harmful.

App development: When developing an app, there is a need to have access to an interface specification that details the services presented by the base computer system to apps (CS-App API). Also, a test bed that simulates essential parts of the ES is necessary to allow early testing of apps.

Federation design: This phase consists of paving the ground for the app development by describing how different actors need to behave in a federation in order to contribute correctly to the desired emergent behavior. This includes definition of roles for the FCs, together with associated rules (or requirements) and communication protocols. Attention should be paid to the mechanisms that lead to emergent behavior so that potentially undesired emergent functions are precluded. In practice, federation design may be carried out implicitly during app development.

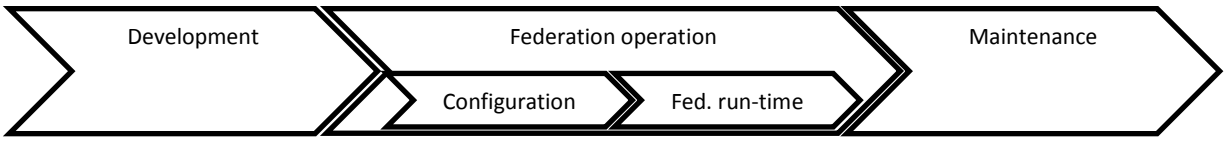


Figure 3. General life-cycle phases of a federation of computer systems.

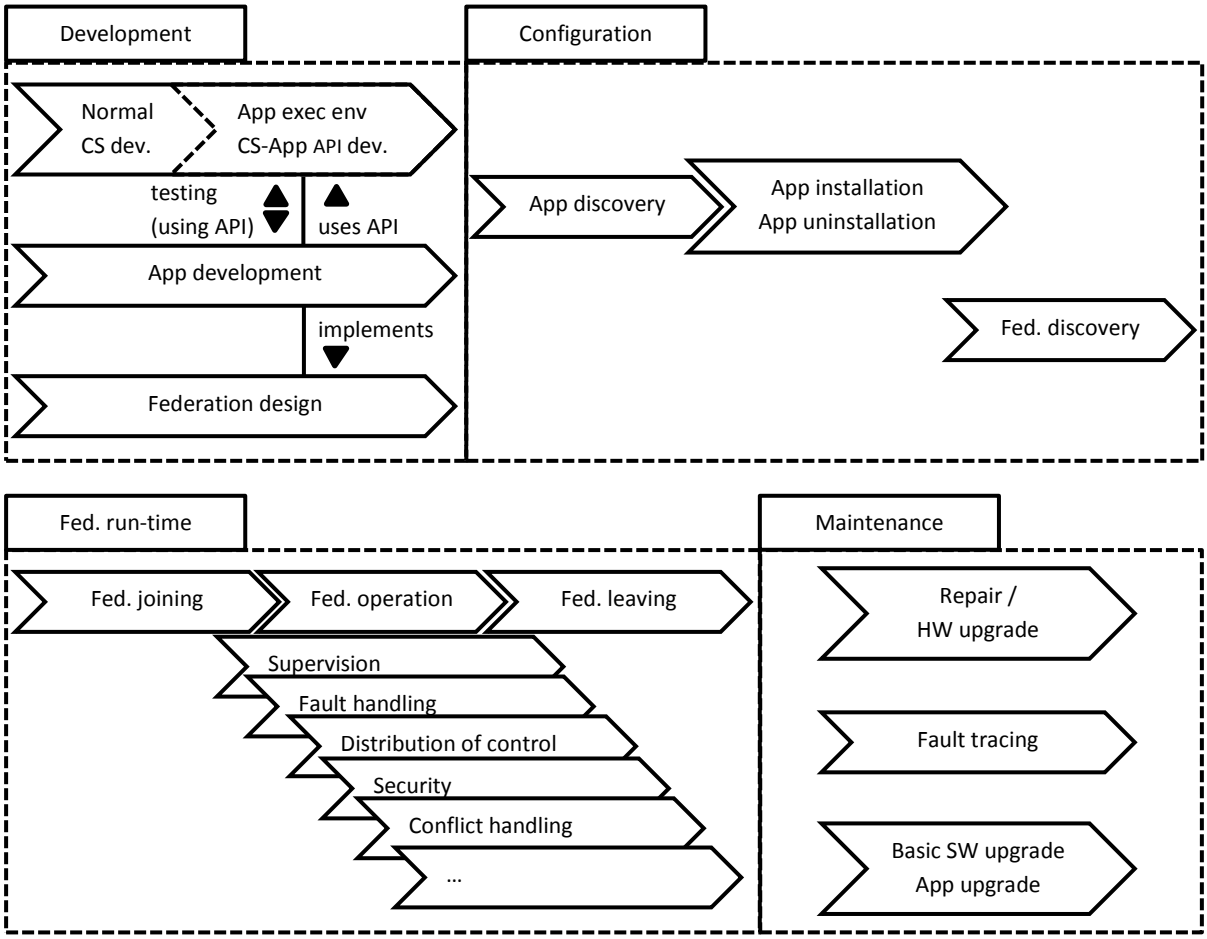


Figure 4. More specific life-cycle phases of a federation of computer systems.

3.2.2 Configuration

App discovery: When the ES is operational, it needs some mechanism for discovering that a suitable app exists. A simple solution to this is user intervention, that pushes the app onto the ES, but other variants are also imaginable. As an example, sometimes additional physical equipment is added to a product (e.g., a new tool to an industrial robot, or new equipment to an agricultural tractor), and these devices need additional control software in the ES. If the added equipment could identify itself to the ES in some manner, it would be possible for the ES to use this identity to initiate a search for appropriate apps on some server. In such a case, security issues need to be considered. Also, safety and usability of the ES may need to be considered at this stage (for example through automatic testing or verification), since they may change as a result of the newly discovered app interfering with the ES resources.

App installation / app uninstallation: Once discovered, the app must be installed. If the ES is distributed, which is often the case, this involves not only downloading the app to the ES, but also finding the appropriate control unit, which involves internal transportation of the app in the ES, and possibly addressing issues. A smooth uninstallation of the app that minimizes the disturbance on the remaining ES functionality, including other installed apps, should also be previewed.

Federation discovery: Once all the relevant software is installed that makes it possible for an ES to become part of a certain type of federations, it must be able to discover when such a federation exists or can be formed, by finding other potential participants.

3.2.3 Federation run-time

Federation joining and leaving: With a potential federation identified, the next step is to join it, and often this will include the need to keep record of some of the other participants. At some point in time, it is also necessary to consider how the ES can leave a federation, and this can be voluntary or not. A voluntary exit is when the federation is no longer relevant to this ES. For example, a federation might be tied to a certain geographical area, and if the ES leaves that area the federation is no longer relevant. A participant could also be forced to leave a federation, for example by the federation supervisor, if it misbehaves.

Federation operation: Between joining and leaving a federation, CS will operate within that federation. This includes a number of run-time operations that may need to be considered, such as fault handling and robustness (incl. classification of fault modes, safe states, fallback mechanisms), operation scheduling on the federation level, cooperation on common tasks and as a result distribution of control, security, conflict handling between different federation components (e.g. resource sharing) but also between the requirements of a component and the rules of the federation, security mechanisms, temporary loss of communication, supervision of the federation behavior, etc.

3.2.4 Maintenance

Repair / HW upgrade: Sometimes, parts of the ES electronics need to be replaced due to malfunctioning, and if such a control unit contains apps and data, it would be expected that the replacement control unit contains the same apps and data. Therefore, some mechanism for keeping track of the content of a control unit is necessary. Since the replacement of a control unit is often caused by the unit being out of order, it may not be possible to store the configuration data in the control unit itself.

A tricky question is whether and how the identity of the unit (or the product that contains it) should be maintained during the upgrade. The history showing how the unit has been behaving may be useful, for example to allow other FCs make correct trust assessments. On the other hand, if a low trust value was previously caused by a malfunctioning HW component, it may be desirable to reset such a history after the repair.

Fault tracing: Most ES contain functionality for diagnostics, in particular to identify problems with hardware. This needs to be extended to also allow diagnostics by apps if they discover problems, and of apps if they function incorrectly in some way. This could also lead to the need of upgrades to the tools used for fault tracing of products in the field.

Software (CS SW + App) upgrade: Occasionally, new versions of an already installed app become available, which could lead to the need for upgrades. This involves discovery and installation as described above, but also removal of the previously installed version. This could lead to a disruption of the functionality of the ES while the upgrade takes place, and also a loss of data. On the other hand, installed apps risk being negatively affected when basic CS software is updated. For example, CS-App API could have been altered which may lead to unforeseen consequences. Another question is how to handle resource scarcity that may arise after a software update.

3.2.5 System life-cycle of the smart traffic lights application

To be able to participate in a smart traffic lights application, each vehicle should be able to accept a velocity reference value. Also, fallback mechanisms are needed to decide whether it is safe to actually follow that reference. If the reference velocity is decided outside of the vehicle, there will be a need to communicate the vehicle position, and preferably also its velocity and destination to the velocity planner. This is the minimal API that should be provided by the vehicle manufacturers.

Probably, it is a good idea to make public the information about the type of communication equipment that each FC carries and which communication protocols it understands. Also, it would be a good idea to supply some sort of testing environment together with the API, for example allowing to simulate how the vehicle's cruise control mechanism react to different inputs.

The federation design is either included implicitly in the implementation of the app software or, which is preferable, performed before the actual app development by some authority that will later certify and/or supervise the correct functioning of the federations. The app development itself could then be outsourced to a third part.

When it comes to configuring FCs so that they can participate in a federation, the final decision about app discovery and installation is probably best left to the owner of the vehicle, unless this functionality is considered to be so basic that it is installed before the vehicle is purchased or during service. However, appropriate installation mechanisms should be in place so that it is simple to correctly install the app in the vehicle. Preferably, from the user's perspective, this should be done at a click of a mouse.

Federation discovery will probably be governed by the app logic, provided that both vehicles and traffic lights contain necessary communication equipment. However, the vehicle users should be able to choose in real time whether or not the vehicle should try to discover and join smart traffic light federations. Correspondingly, the vehicle users should be able to leave a federation instantaneously, which is important for example if the velocity planning logic works poorly for some reason. FCs that are more central to the emergent functionality, such as central planners and traffic lights, may have some restrictions on when and how they may leave a federation.

During operation, safety and security considerations are probably best distributed to the individual FCs. However, conflicts between the FCs, for example conflicts between the vehicles on intersecting roads, may benefit from being handled in a centralized way. Also, the vehicles could detect unconnected vehicles and communicate such information to the planners and the traffic lights.

It might be a good idea to record that the smart traffic lights app has been installed in a vehicle, so that it can be automatically re-installed if the control unit where it was located is replaced. Such

information can for example be stored on a dedicated control unit internally in each vehicle, or using the cloud concept.

Stationary apps, i.e. apps located in planners and traffic lights can only recommend the reference velocity values to the vehicles. If we assume that the vehicles contain diagnostic mechanisms to avoid following unrealistic or dangerous recommendations, faults in the stationary apps seem less serious and would only lead to the dissolution of federations since vehicles only stop listening to the velocity recommendations. With this in mind, stationary apps could safely be updated during the federation operation.

When it comes to the vehicle apps, unexpected internal interactions or resource scarcity due to software update, may lead to more significant consequences. Thus, both apps and basic software in vehicles should only be updated when the vehicles are standing still, and preferably in a service situation, where common driving scenarios can be tested, either on a test drive or in a simulation environment.

3.3 Software ecosystems

Not surprisingly, development processes, and more generally software ecosystems, will also be affected by the increased interaction between ESs. During one of the workshops, the distribution of responsibility between different actors (such as producers, users, authorities, etc.) was discussed. To structure the discussion somewhat, different ownership scenarios that exist today were used. In the following figures, each actor will be represented by a non-yellow color and an identification letter, while yellow-colored empty blocks represent the actual products (P), consisting of embedded systems (ES) with associated environments (Env.), apps, computing devices, and servers, see for example Figure 5.

3.3.1 Apple-type software ecosystems

The first scenario was inspired by Apple's tight grip of its products and services. In Figure 5, the distribution of responsibilities that Apple dictates is demonstrated. In Apple's case, the server side (the App Store) is developed, maintained, and supervised by Apple itself. The same goes for the development of hardware (e.g. iPhones) and associated basic software. Apple may also develop some additional software after the hardware deployment.

The other actors in the Apple scenario are users and owners of the products, as well as 3rd party producers of apps. While these actors have some freedom in how to use and even affect the end product, Apple maintains the highest authority by certifying apps before they can reach the App Store. In other words, using the concepts defined in Section 3.1, Apple acts as an off-line federation supervisor with an influence on the app developers, which are the actual federation designers. Online supervision of interactions that occur between different systems is absent even in this relatively controlled software ecosystem since even though Apple may withdraw a malicious app from the App Store, it is generally not interested in detecting on the fly if erroneous information is being shared.

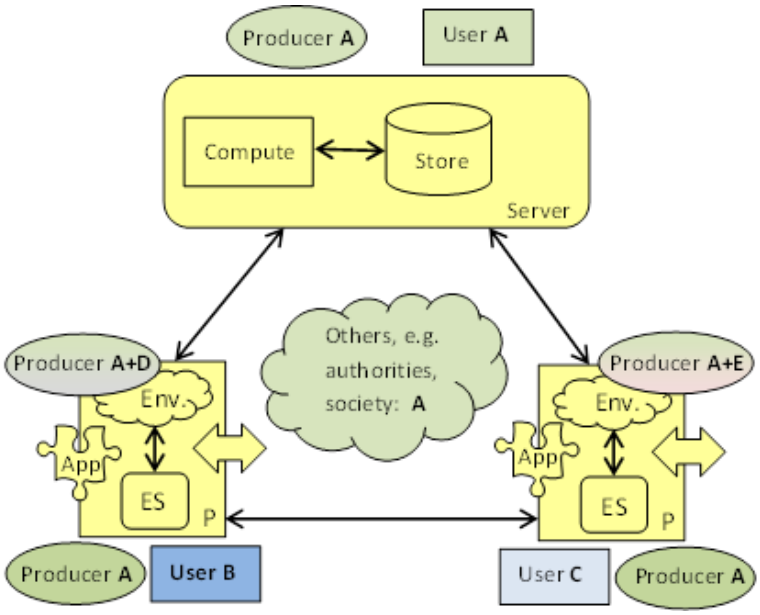


Figure 5. Apple's model of the distribution of development responsibility.

3.3.2 Google-type software ecosystems

The other scenario, inspired by the Android approach, is somewhat opposing, although it comes from the same domain, see Figure 6. Evidently, the number of actors is higher in this scenario, even though there is still a single responsible (Google) for defining the rules of usage and upgrade (e.g. through operating system (OS) development), as well as for maintaining the server side (Android Market). However, hardware development is largely left to other companies, such as for example Samsung or HTC. Likewise, software development is not Google's concern, except for the OS development of course.

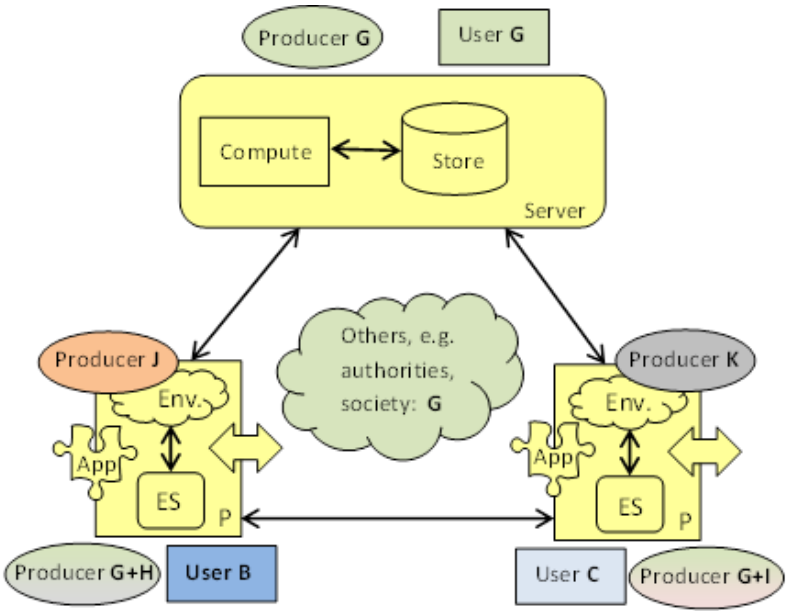


Figure 6. Google's model of the distribution of development responsibility.

Federation supervision is left to the users of the federation components. Although the Android Market requires certificates, they are allowed to be self-signed by the app developers and are only used to identify the author of the application. The idea is that the possibility to trace the identity of each app developer will ultimately lead to ad-hoc federation supervision through some kind of reputation models within the user community.

3.3.3 Industrial automation type software ecosystems

A third scenario comes from the world of industrial automation, Figure 7. Here, there are only two actors, the producer and the user of the automation equipment. Normally, a single producer delivers all necessary equipment (e.g. manufacturing robots and their control devices) to a single user. The user will decide on how to actually configure and use the equipment, including the software for controlling its manufacturing processes in an appropriate way. In fact, the actual implementation of the software is often laid out to a 3rd party line builder company. However, the responsibility for the requirements is still owned by the user of the equipment. The framework regulations are often defined and maintained by the producer in cooperation with the user's needs.

Thus, the federation design can be said to be done jointly by the user and the producer, while the actual functionality of the federation and its conformity with the requirements is most often supervised by the user.

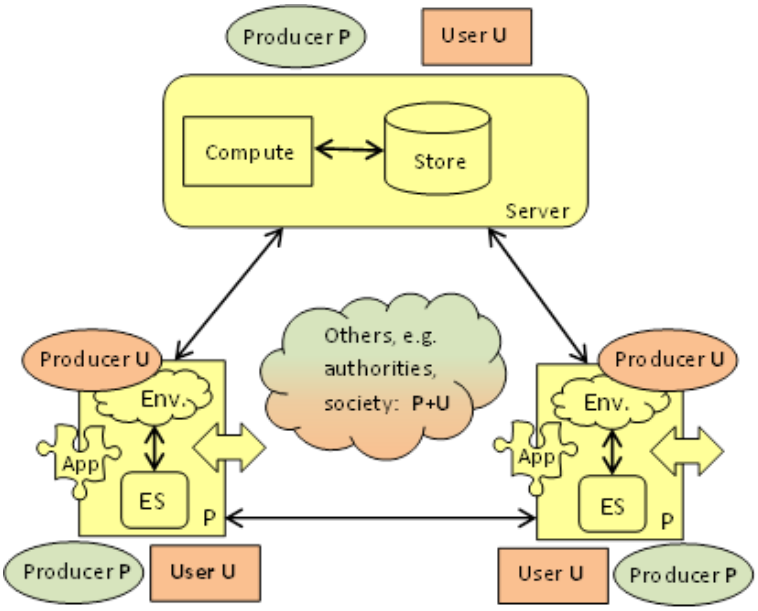


Figure 7. The distribution of development responsibility in industrial automation applications.

3.3.4 Smart farm type software ecosystems

The fourth software ecosystem scenario comes from the agricultural world. Here, a smart farm was imagined, where the farmer owns or rents a number of machines from different manufacturers that may be upgraded after their deployment. Today, hardware upgrades in such machines are routinely done by 3rd party companies that adapt or re-build for example trucks for their user's aims. Here, we also consider the possibility of 3rd party software development companies, possibly certified ones,

which help the farmer to adapt the functionality of the agricultural machines through software add-ons, similarly to how things work with hardware re-construction.

In Figure 8, the identified actors are illustrated. There is a single user (the farmer), that sets the framework for its farm, while also respecting some rules, which are often decided politically. Apart from this, the openness of the scenario is very high, with a possibility of every piece of hardware and software being delivered by a different producer. The federation design may include such actors as machine providers, software developers, farmers, and politicians. The high openness of such an ecosystem, without any single dedicated federation designer, assuming that the farmer himself is not a full-time computer scientist, accentuates the question of who is responsible for the supervision of the emergent functionality and for making different pieces of equipment fit together.

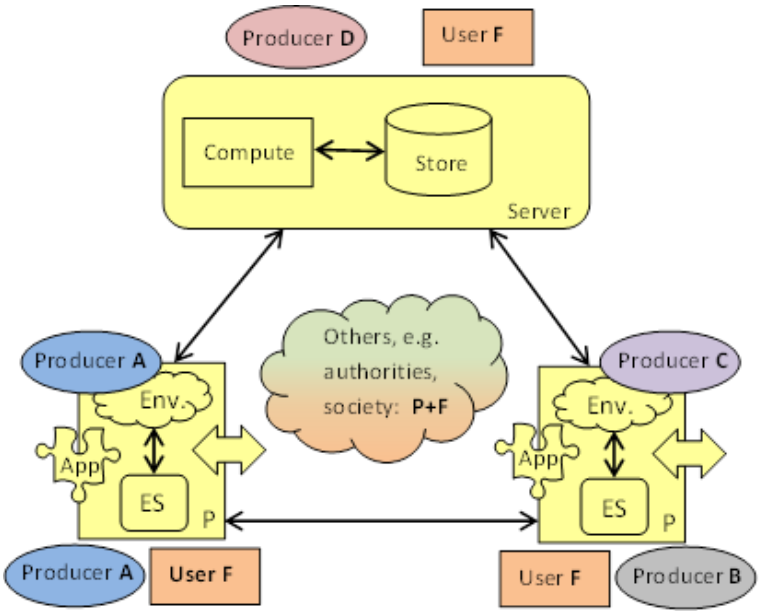


Figure 8. The distribution of development responsibility in a smart farm.

3.3.5 Software ecosystems of the smart traffic lights application

When it comes to the smart traffic lights application, it soon becomes clear that the software ecosystem situation is highly heterogeneous. The number of equipment producers is high (vehicle manufacturers, traffic light manufacturers, planner GPCS manufacturers), while the number of different users is as high as the number of vehicles in the federation. The federation rules may be designed or influenced by several interested parties, such as road administration authorities (RAA), vehicle manufacturers, and driver associations. The federation supervision and certification of apps could be done by the RAA, but it is also imaginable that the vehicle manufacturers take a tighter control over apps, only allowing installation of apps that have been certified by them. There may also be competitive apps that provide the same main functionality.

In fact, heterogeneity may be one of the main challenges for this kind of applications. Standards and well defined ecosystem structures are necessary to cope with this situation.

3.4 Organizational structure

When designing a federation, it is crucial to decide on its organizational structure. This non-trivial decision affects most other functionality, such as planning, supervision, data analysis, data storage, fault handling, etc., with consequences to several federation characteristics, such as failure risks and effects, distribution of processing load, security, etc. In turn, the choice of an organizational structure may be influenced by a number of things, such as the available technology, whether there are some common SoS-level goals, robustness requirements, how much control the individual federation components are prepared to cede, stakeholder requirements and business models, etc. The complexity is further increased by the fact that different organizational structures may be chosen for different aspects of the federation functionality. For example, data management and planning could be organized in different fashions. In conclusion, the organizational design of federations of embedded systems seems to be an important question for future research.

In [27], some organizational paradigms from the multi-agent research community are surveyed. These are summarized in the following list:

- *Hierarchies* – a tree-like structure, where agents higher in the tree have a more global view. In its strictest sense, there are no interactions across the tree. Data travels up the tree, control travels down. Often suitable for divide-and-conquer approaches. May hierarchies suffer from single points of failure and bottleneck effects.
- *Holarchies* – quite similar to hierarchies, the differences being more cross-tree interactions and more local autonomy (as in flat hierarchies). Also, there is an explicit notion of *holons*, or "systems of" subsystems or subcomponents. Generally, individual holons decide how to implement tasks that they are given. Flat holarchies resemble federations. As with hierarchies, holarchies are suitable for decomposition.
- *Coalitions* – any subset of an agent set is a potential coalition, as long as every participant in such a subset gains some advantage from participating. Very flat structures, generally goal-directed and short-lived. Every participant is in for its own good.
- *Teams* – flat structures, as coalitions, but agents try to maximize not their own but a common goal. Type and pattern of interactions are unspecified which includes most other organizations. Distinguishing features are explicit representations of the team goal, such as joint mental state and the ability to reason about the consequences of the team behavior.
- *Congregations* – also a flat structure. Long-lived formations of agents with similar or complementary characteristics with the goal of facilitating the search for partner agents. Goals can change over time, but every agent should have a stable set of characteristics and requirements. Are formed to reduce interactions (since it is easier to find appropriate partner agents), thus there are generally no interactions between different congregations.
- *Societies* – open, heterogeneous systems or rather operating environments, with the central authority replaced by a set of rules (sometimes referred to as social laws) and protocols that specify the interactions within a society. Normally, the society rules need to strike a balance between the objectives of different participant groups. Within a society, agents can be sub-organized into other organizations.
- *Federations* – groups of agents which have ceded some amount of autonomy to a single delegate which represents these groups (intermediary). Interactions always pass through the intermediary, who must be able to understand both its federation member and the

intermediaries of other federations. Examples of intermediaries: facilitators, brokers, mediators, translators, embassy agents. Note that the concept of federations differs from the one employed in our setting.

- *Markets* –consist of typically competitive participants, such as buyers, sellers, and sometimes third party agents named auctioneers. Apparently similar to federations, although agents do not cede any authority.
- *Matrix organizations* – differently from many other structures, market organizations allow every agent to be directed by several managers or peers. This creates interesting possibilities to model conflicting interests.
- *Compound organizations* – combinations of the other types, for example hierarchic control organization and coalition-like data management.

The above organizational structures are somewhat overlapping, and yet quite well defined, with their own drawbacks and benefits. It remains to be seen which of these structures may be useful for FES applications.

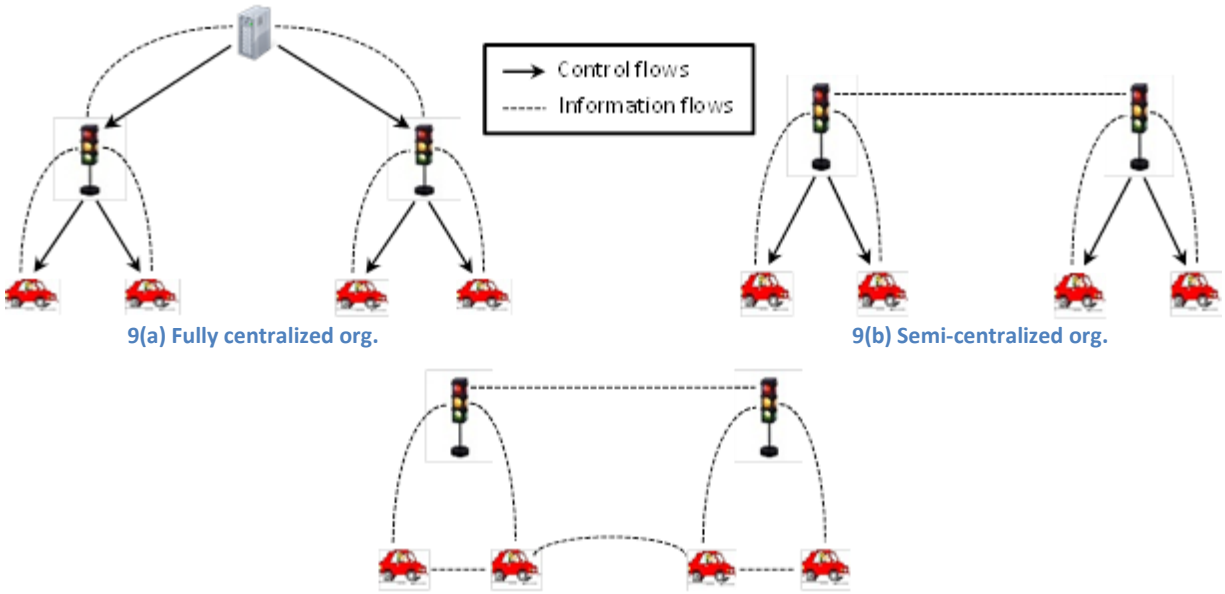
During the workshops, organizational structures were discussed on a higher abstraction level, distinguishing between the interrelated concepts of *flat* (FCs communicate directly with each other) vs. *hierarchical* communication, and *centralized* vs. *decentralized* planning. Flat organizations generally lead to a higher burden on communication networks, while the hierarchical ones have the drawback of longer chains of message passing and thus longer communication delays. The centralized planning risks creating single points of failure, while decentralized planning may sacrifice the overall predictability within a federation.

3.4.1 Organizational structure of the smart traffic lights application

At one of the workshops, three types of organizational structures for the smart traffic lights application were discussed, fully centralized, semi-centralized, and fully decentralized, Figure 9.

In the first scenario, see Figure 9(a), the organization is assumed to be centralized hierarchical. All vehicles and traffic lights within some neighborhood communicate with and are directed by one central planner. There might be several hierarchical levels, with the decisions of regional planners being influenced by higher-level planners. Some benefits of this approach are the simplicity of the idea and a rather good predictability of the system behavior, assuming that the vehicles and traffic lights follow the plans that are made for them. Some drawbacks are the computational and communicational burdens on the planners, as well as the sensitivity to planner failures. Uncertainties are created by non-compliant or non-connected vehicles, which may lead to frequent re-planning of whole regions.

The second approach, see Figure 9(b), consists of regional planners, physically located either in dedicated GPCSs, traffic lights, or even in (some) vehicles. The planners interact with each other, partly to make better decisions, and partly to promote robustness in the federation by supporting each other in case of high processing loads or system failures. Of course, such support activity presupposes that protocols for redistribution of authority have been foreseen by the federation designer. This solution seems more robust than the first one. However, the need for re-planning of regions due to uncertainties is still quite high.



9(a) Fully centralized org.

9(b) Semi-centralized org.

9(c) Decentralized org.

Figure 9. Three conceivable organizational structures for the smart traffic lights application.

The third organizational structure that was discussed during the workshops was assumed to be fully decentralized and flat in the sense that the planning functionality was distributed to each vehicle and traffic light in the federation, see Figure 9(c). Each vehicle receives planned green light time periods from the nearest traffic light along its path (or a number of candidate paths) together with that traffic light's coordinates. This information should be sufficient for the vehicle to decide its own preferred speed and even choose the fastest path to its destination. Vehicles may exchange some information directly with each other, for example regarding traffic jams, to avoid congesting the communication capacities of the traffic lights. The vehicle velocities and directions are communicated back to the traffic lights, allowing them to adapt their green light periods based on that information.

The decentralized solution seems to be the most robust since it avoids single points of failure, as long as communication networks are up and running. Also, in case of information uncertainties, it may suffice to only re-plan those vehicles that are directly affected by the uncertainties. Some drawbacks are the processing load that is placed on the vehicles, which may be detrimental to the rest of the vehicle's functionality; the higher interaction complexity due to the need to negotiate between different vehicle-based planners leading to a higher communication burden; and the higher difficulty in predicting the emergent federation behavior. Also, unless there is a central authentication authority that monitors new federation members, security may be jeopardized.

3.5 Reference architecture

In this section, the functionality of federations of embedded systems that came up during the workshop discussions will be summarized in an architectural model. In Figure 10, such a model is outlined in a standard UML-notation. The model consists of three main parts that contain *planning*, *supervision*, and *data management* functionality.

3.5.1 Planning

Generally, in our setting the information that is exchanged within a federation aids ESs to plan their actions in a more informed way, thereby adapting to a changing environment and improving on existing or offering new services to the user. In other words, action *planning* or *coordination* between ESs is the key functionality in a federation. Depending on how control flows are organized within a federation, it may take on different forms.

One case resembles classical *feedback control*, with an external CS providing some reference values to the FCs. In such a case, the planner CS plays the role of an outer loop controller, while the other FCs are responsible for adjusting their internal controllers to follow the reference value. However, normally FCs should maintain the possibility of refusing to follow a reference value, not least for safety reasons. In another planning scenario, FCs retain a higher level of autonomy and a higher responsibility for their own control actions, while a central planner only *schedules* the desired order and possibly time limits for the actions within a federation or a part of a federation. Finally, in its most decentralized form, the planning functionality is left to the individual FCs while the coordination between them is done on a *negotiation* basis.

3.5.2 Supervision

Supervision is another important functionality that should be present in a federation, especially if negotiation based planning is used, since faults will eventually occur both in the individual FCs and due to the sometimes unpredictable interactions between them. In order for the federations to be robust, faults or security attacks must be detected by some sort of *monitoring* mechanism and countermeasures should be taken. Monitoring will generally need *environment models* as reference lines. These models could be non-deterministic and even include elements of subjectivity, for example it could include perceived *trust models* of the exchanged signals.

As far as possible, fault states of both the federation and the individual FCs should be previewed during the design phase, together with the corresponding safe states towards which the systems would be directed by some *fault handling* mechanisms, preferably in a way that affects the emergent functionality of the federation as little as possible (graceful degradation). Conflicts will most probably also occur and will normally not be appreciated by the individual FCs and their users, even though they may hold the potential for the evolution of the federation as a whole. Thus, conflicts should be prepared for in advance, detected during runtime, and prevented from affecting the overall behavior too significantly. Due to the apparent similarities, *conflict handling* is considered as a special case of fault handling.

3.5.3 Data management

The third functionality block is related to the management of data that is collected and exchanged within a federation. This includes a number of design questions, such as which data should be kept secret and which should be shared; where to *aggregate* the data; where and how long should the data be *stored*; how to perform *data analysis* in the most efficient way, etc.

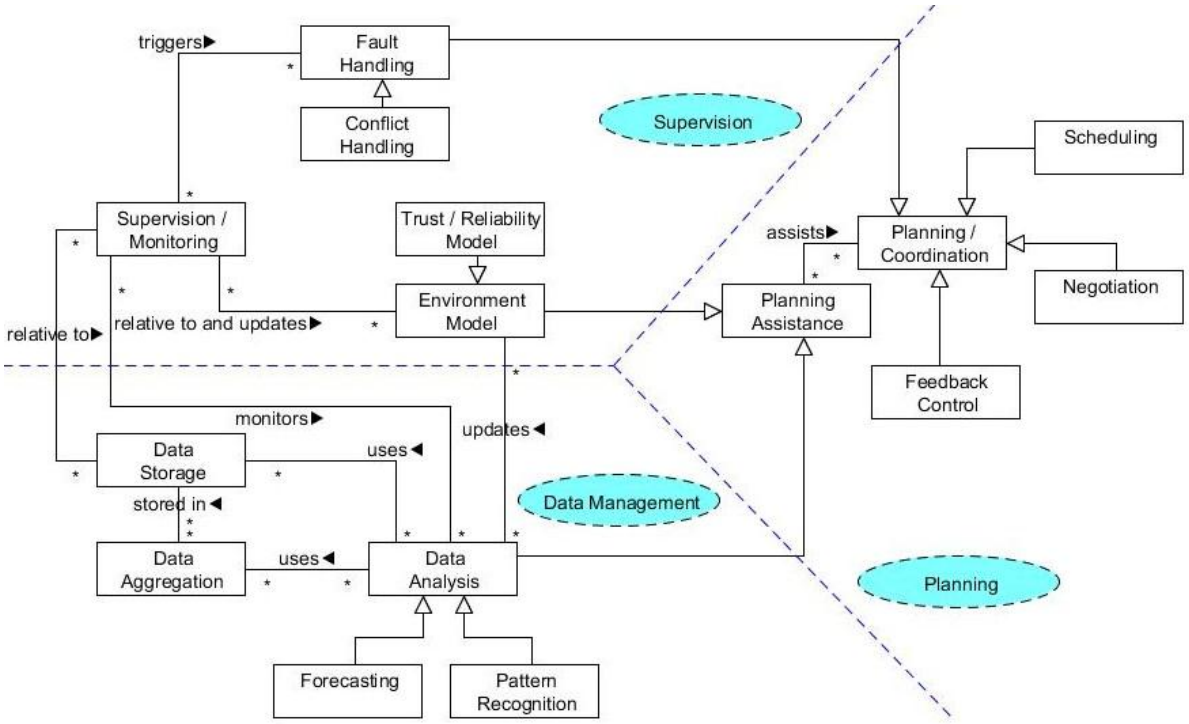


Figure 10. The reference architecture of common mechanisms in a federation of computer systems, using UML-notation.

Typical analysis procedures will include *forecasting*, often using some sort of *pattern recognition* with respect to historical data and/or some model of the FC environment, in order to provide *assistance* to the *planning* mechanisms. The data analysis also forms a basis for the supervision activity, both in form of forecasting, current data patterns, historical data, and updates to the environment models that the supervision mechanisms rely on.

3.5.4 Alphabetical functionality list

In the following, federation functionalities are listed in the alphabetical order and defined more properly.

Conflict handling: Conflicts between FCs are probably unavoidable in open, flexible systems of systems. This need not necessarily be negative since conflicts sometimes lead to a surge in development and evolution of systems. However, conflicts generally lead to deviation from the designed emergent functionality, and need to be handled properly to avoid non-desired federation behavior. In this aspect, conflicts can be seen as a special case of faults.

Conflicts can occur both between different requirements of the components, and their execution plans. If a conflict occurs between federation components that are subordinate to the same scheduler, chances are good that such a conflict can be resolved by rescheduling a part of SoS. In other case, negotiation may be the only remedy to resolve or mitigate a conflict. Finally, if a federation component is in conflict with the rules of the SoS, its future within the SoS is at risk. However, repeated rule violations should be reported for future upgrades of the federation rules.

Data aggregation: Federation components (especially if they are embedded systems) will often be able to collect some sort of data. Such data may be shared between federation components and

stored in an appropriate place. Also, it may often be the case that the data is processed before sharing.

Data analysis: The aggregated data will often be analyzed in some way to aid the planning procedure. Typically, data may be analyzed to detect behavior patterns or to forecast future trends. As a result of the data analysis, environmental models may need to be updated. For example, a gradual slow-down of vehicle speeds may be recognized as a traffic jam in creation. On the other hand, if the environmental model of a SoS includes the information about a traffic jam at 9.00 every Tuesday, this should be reconsidered (and preferably automatically updated) if the above build-up pattern is no longer present on Tuesdays, for example after a road reconstruction.

Data storage: The aggregated data will sometimes be stored, either locally in a federation component or in a more central location. It can then be used for future data analysis, diagnostic monitoring, etc.

Environment model: A model of the environment that affects a federation component. Such a model can be present from the deployment of the component (defined in its basic software) or updated during its operation, for example as a result of data analysis or software update. The environment model can be used as a reference when monitoring the reliability of incoming signals. For example, a reference velocity of 200 km/h on a Swedish highway may be judged as unreliable.

Fault handling: If a supervision mechanism detects a fault, it should be handled, thus affecting the planning procedure. Fault handling will often be based on a description of safe states and mechanisms for directing the SoS or some of its parts to these states. A common type of fault is when a federation component stops receiving an expected signal or is receiving an unreliable (low trust value) signal. In such a case, a local fallback mechanism (safe state) should be foreseen. Conversely, faults can be handled globally, for example by expelling an unreliable component from the SoS or by redistributing the functionality of a failed component to other federation components. Faults should normally be reported either to the affected FCs or directly to the federation supervisor.

Feedback control: In many applications, a central planner will request FCs to follow some reference value. This is quite similar to classical nested control, with the planning mechanism playing the role of an outer controller, while FCs are internally responsible for the functionality of the inner control loop making sure that the outer loop reference is followed. Thus, the feedback control term will be used in such situations, even though there is a difference with the classical control since FCs generally have more autonomy and may decide on their own not to follow the reference value.

Forecasting: Prediction of the future state of affairs in a SoS (states of the federation components, interchanged signals, etc.) on the basis of the aggregated data, in order to assist the planning procedure. Historical (stored) data may also be used for better forecasts.

Negotiation: If the organizational structure of a federation is fully decentralized, as a consequence there is no central authority to coordinate the federation components. Instead, coordination is done through some sort of negotiation or handshaking. This does not necessarily mean that the federation components are equipped with artificial intelligence (AI), but rather that the interaction protocols are well defined. For example, a simple rule stating that the FC with the lowest IP-number should be granted access first to a shared communication media is considered as a negotiation protocol. After

all, in that case each FC needs to supply its IP-address and wait for the others to accept it as a "winner" or present it with a counterexample, which is quite similar to the common understanding of negotiation, only with the AI part removed. In fact, in our setting, the level of AI will be kept at a minimum in order to keep the federation behavior as predictable as possible.

Pattern recognition: Recognition of familiar patterns in the aggregated data, allowing to draw conclusions about the current state of affairs in a SoS (e.g. states of the federation components). Familiar patterns may be either pre-defined or accumulated during SoS operation.

Planning / coordination: Planning or coordination is central for the cooperation within a federation. This can be carried out in different ways, e.g. fully centralized, fully decentralized, or a mix of both, depending on the chosen organizational structure for the control flow. Even if planning is not fully centralized, some sort of negotiation mechanisms for coordinating interactions between the federation components will still be needed. Planning includes adaptation to changed environment and federation composition. It is also important to plan in such a way that a balance is achieved between the federation goals and the needs and characteristics of the individual FCs. For example, safety critical functionality in FCs should normally not be jeopardized by emergent functions.

Planning assistance: Mechanisms that, in one way or another, are used to assist or influence the planning process.

Scheduling: A planning mechanism that requests the federation components to execute certain functionality in a certain order. Time information for the start and/or end of execution may be included in such a request. This presupposes some kind of hierarchic structure with the scheduler standing above the scheduled federation components. However, for security reasons the components will generally have the final say about whether to execute such requests or not. A special case of scheduling is resource allocation.

Supervision / monitoring: The aggregated and processed (analyzed) data should be monitored in order to detect faults. Normally, this will be done either with respect to stored data patterns or to some environment model. If a fault is detected, appropriate fault handling mechanism should be triggered.

Trust / reliability model: A special case of the environment model construct that models the perceived reliability of incoming signals. In some applications, it will be natural to update the reliability models when faults are detected. Again, these models can be implemented either in a local or a global way. In case of local reliability models, a mechanism for sharing known trust values between the federation components may be useful.

3.5.5 Reference architecture of the smart traffic lights application

In this section, the proposed reference architecture for SoS functionality will be discussed with respect to the smart traffic lights application, shortly described in Section 2.1.2.

The type of planning functionality is dependent on the choice of organizational structure. Here, we return to the three organizational types for the smart traffic lights example that were discussed earlier, see Section 3.4.1. In the fully centralized scenario, planning becomes a combination of scheduling and feedback control, where the planner first tries to find an optimal intersection

schedule and then, based on this schedule, communicates reference values for velocities to the vehicles.

Similar situation occurs in the semi-centralized case, where scheduling is performed regionally. The difference to the fully centralized scenario is that scheduling is distributed both geographically and to several hierarchical levels. This leads to smaller scheduling problems locally, but a more complex logic in order to coordinate, and if necessary balance processing loads, between the regional planners.

In the decentralized organization, planning functionality is mainly in the form of negotiation between the FCs, even though internally each vehicle and traffic light will need both scheduling and control mechanisms, for example to calculate and follow its own preferred velocity. An open question is whether there is a need for mediators to facilitate negotiation between FCs.

To simplify the following discussion, it is assumed further that the smart traffic lights application has been chosen to be organized in a semi-centralized fashion. Regional planners are supposed to be located at each traffic light, while one central meta-planner is responsible for balancing the traffic load on a city scale.

Recall that three roles were identified for the smart traffic lights application: “vehicle”, “traffic light”, and “planner”. Each role can be described by a collection of rules or executable procedures. For example, a vehicle can join the federation and play the “vehicle” role if it accepts the following rules associated with this role:

- Send the information on a regular basis about its position, speed, and possibly destination to the planner;
- Listen for the velocity recommendations from the planner and use them as reference values in the internal velocity control loops (note that it is the responsibility of the vehicle and not of the SoS to make sure that this can be done safely);

The “traffic light” role adheres to the following rules:

- Send the information about the time for planned green slots to the planner;
- Send the information about the mean speed of the passing traffic to the planner (optional);
- Listen for the information about recommended green slot times from the planner and adjust these slots accordingly;

The “planner“ role is governed by the following rules:

- Listen for the information about the times for planned green slots of the traffic lights
- Listen for the information about vehicle speeds, positions, and possibly directions;
- Calculate the optimal green light periods (with respect to the overall traffic flow) and transmit this information to the concerned traffic lights;
- Plan the velocities of connected vehicles and transmit these plans to the vehicles.

Note that the above execution rules only define the behavior of the federation components on the SoS-level, i.e. with respect to the other FCs. Naturally, there will also be internal procedures, tightly related to the above rules. For example, vehicles will have to decide if it is reasonable to follow speed

recommendations and only adjust their speeds if the decision is positive. In practice, this can be easily achieved by passing the speed reference value to a cruise control mechanism. In modern cars, such mechanisms include radar- and/or video-based safety systems that override cruise control if necessary to avoid collisions with slower moving vehicles.

This is a typical example of the use of an environment model that affects planning or execution of a SoS process. Another model could include speed limit information for each vehicle location. A third example on the same topic is the trust to the validity of the requested reference speed. Suppose for example that the planner has a history of incorrectly predicting the optimal speed at rush hours, within a margin of ±10 km/h. Ideally, the vehicle app should account for this. For example, it could adjust the reference speed (by at most 10 km/h) after comparing it with historical data (data storage) and/or radar data (environmental model) if the trust value of the planner is low (as it would be at rush hours).

The data comparison procedure itself (analysis) may contain elements of pattern recognition and/or forecasting. For example, if a vehicle realizes during its data aggregation that the pattern of surrounding vehicles resembles a build-up of a traffic jam, it may be more skeptical to the planner's information, especially if the planner is located a bit away. Of course, a smarter app would forward such information to the planner, thus assisting it in its planning.

It seems natural to handle vehicle related faults internally. The same goes for the hardware faults in the planner FCs. The emergent federation behavior, mostly caused by the planner's decisions together with unpredictable vehicle behavior and the existence of non-connected vehicles, needs to be supervised on a global level. The information from the vehicles about their recommended and actual velocities, as well as their radar and video signals of their surrounding environments, should be processed either in the planners on in some dedicated supervision GPCSs. For example, frequent vehicle speed deviations from the requests of a certain planner may serve as an alarm signal to watch closer at that planner's status. If all planners frequently change their recommendations (instability), it might indicate the need to revise the app that regulates the emergent functionality. Finally, this application seems to provide good ground for conflicts between intersecting traffic streams, especially if these streams contain vehicles with different priorities, such as ambulances, which must be accounted for during planning.

4 Research challenges

During the workshops, a number of research challenges of both technical and non-technical nature were discussed. In this chapter, they are shortly presented.

4.1 Dependability

Questions of dependability, in one way or another, were one of the subjects that received quite some interest from the workshop participants. Dependability is sometimes defined as the ability to deliver service that can justifiably be trusted [28] and consists of several attributes that vary somewhat depending on the source. In [28], dependability is said to be composed of:

- *Availability*: readiness for correct service
- *Integrity*: absence of improper system alteration
- *Maintainability*: ability to undergo modifications and repairs
- *Reliability*: continuity of correct service
- *Safety*: absence of catastrophic consequences on the user(s) and the environment

In the same position paper [28], *security* is defined as a separate but related concept to dependability and composed of availability, integrity, and *confidentiality*, i.e. the absence of unauthorized disclosure of information, also known as *privacy*. In other sources, security is a part of dependability and will be treated as such in this work.

A closely related concept to dependability is *resilience*, often defined as the speed of recovery from a degraded system state.

In the following, the highlighted concepts will be briefly discussed. Generally, it is important to consider these aspects both on the individual FC level and on the federation level.

4.1.1 Availability

Availability is measured as the fraction of time that the system is ready to provide its service. Robustness is crucial for high availability, while in applications that suffer from single points of failure, availability is obviously at risk. Thus for this sake, control mechanisms and well defined protocols for redistribution of responsibilities, are needed, together with efficient diagnosis algorithms that rapidly isolate malfunctioning or overloaded equipment.

At the FC level, it is crucial to make sure that the availability of safety or business critical functionality of the individual products is not compromised by their participation in a federation. Again, clear rules for what interactions may be acceptable and when are needed. For example, is it acceptable to let a route planning algorithm in a vehicle make use of 80% of the available processor capacity, even when it is done for a short period of time and no other function is currently running in the vehicle? What if this vehicle acts as information provider to other FCs, such as emergency vehicles? Such questions amount to the challenge of striking the balance between SoS and FC level functionality, or even between the functionalities of different federations.

A closely related concept to the availability is *resilience*, often defined as the speed of recovery from a degraded system state. Construction of safe-proof systems is highly unrealistic, even in the case of closed proprietary systems. Thus, the notion of resilient design is highly important in SoS applications, in order to anticipate system failures so that the functionality is degraded as little as possible and recovered as fast as possible. A potential complication is the trade-off between SoS

level and FC level functionality. For example, it might be wise for an FC to leave a malfunctioning federation, which could lead to failure escalation in the federation functionality.

4.1.2 Integrity

Integrity stands for the absence of improper or even adversary system alterations. The question of adversary alteration falls under the security discussion, while improper system alteration is a clear risk factor in the federations of ESs that are treated in this work, being large-scale, open, and flexible systems, with a potentially high number of stakeholders. The scale and flexibility of federated systems lead to difficulties in predicting the emergent federation behavior, as well as the behavior of individual FCs, while openness makes it hard to control the quality of software in a federation.

Some sort of certification mechanisms will be needed, possibly relying on testing and formal verification technologies. Certification should take into account how possible interactions between different FCs, with their variety of different federation apps, affect both the SoS- and the system-level behavior.

Another kind of certification that may be useful is to approve or rate the app developer companies, for example similarly to how it is done in the heavy vehicle industry, where certified reconstruction companies adapt original trucks and trailers to the specific needs of their customers.

4.1.3 Maintainability

Maintainability is the measure of the mean time that is needed to repair a system, thus having a direct impact on the system's resilience. Ideally, each FC should be easily maintainable, leading to high resilience of the federation as a whole. However, this might be difficult to achieve, especially in federations that consist of a large number of heterogeneous components.

Instead, low maintainability may be countered by high robustness, for example through redistribution of responsibilities of a malfunctioning FC within the federation. Conversely, in cases of low robustness, or more specifically in cases of critical FCs, the maintainability should be accounted for during the federation design.

4.1.4 Reliability and trust

Reliability is the measure of the mean time to failure. Ideally, federations should be designed in such a way that this time is close to infinity. In other words, federations of unreliable components should be designed to be reliable, for example by letting FCs to take over each other's responsibilities in case of failures. Of course, in real life, this might be tricky, either due to different types of equipment not being able to play the same roles in a federation, or due to some FC-internal functionality that must be provided, actualizing the question of balance between FC and SoS availability.

The reliability is related to the concept of trustworthiness that may be important in federations of heterogeneous FCs. In the end, each FC will have to decide whether or not to trust a signal value that it receives from some other FC. Trust may be affected by the reliability measure, a history of frauds, geographic location, etc. How trust should be represented, stored, and updated are interesting questions that merit further investigation.

4.1.5 Safety

Safety is the absence of catastrophic failures in a system. Much discussion was devoted to fault handling during the workshops, i.e. how to prevent faults from escalating into severe failures. Since

the number of interacting systems may be quite high, it is important to isolate the root cause of the problem and to avoid that it propagates to the other FCs.

In other words, efficient diagnosis of highly communicative large-scale systems of systems that arise in our setting is needed. On the one side, the scale of the problem makes the algorithmic complexity high, while on the other side, the extensive information exchange that is expected within a federation may facilitate the diagnosis.

When it comes to the federation design, an obvious safety measure is to try to avoid single points of failure. Another idea is to define fault states for each FC and to have pre-designed (graceful) degradation mechanisms for bringing the FCs into some safe states. A challenge is to develop mechanisms for efficient composition of FC fault and safe state descriptions into federation global models. This is important in order to capture fault conditions that arise due to conflicts between FCs that are not detectable when considering FCs in isolation.

Due to the federation behavior being not fully predictable, some sort of global supervision mechanisms will be needed. An important goal of such federation supervision is to focus on the conflict induced faults and try to resolve or mitigate them. Additionally, in the end each FC must be responsible for its own safety. Thus, if FCs are affected by faults or conflicts in other FCs, and these faults have not been resolved after a period of time, these FCs should have fallback mechanisms allowing them to leave the federation in a safe way. Also, FCs should communicate about their own faults to the rest of the federation, even though they might be unwilling to do so in some situations, for example if this would risk affecting their trust values.

4.1.6 Privacy

Privacy is an important aspect in most, not to say all, applications of federated ESs. If the information is used for malicious attacks, questions of privacy immediately translate into security considerations. Privacy challenges embrace questions of how to separate critical and non-critical information, which information is critical to whom, how to aggregate and store data without compromising privacy, etc. These are clearly non-trivial and important questions.

4.1.7 Security

Opening up an ES for external communication is connected to a risk of also creating security hazards, and when the external communication is combined with the ability to also install new software in the device, additional vulnerabilities to external attacks could surface. Such weaknesses have been shown both in real life, e.g. by the Stuxnet worm for industrial control systems [29], and in academic experiments with automotive systems [30]. Although security has not been the primary concern for this investigation, it is clear that these issues cannot be neglected. A few measures that can be taken to reduce the potential security risks with installing malware are the following:

- *Execute apps in a sandbox environment*. This includes running the apps on virtual machines embedded within the normal control units in the ES, and running all app communication between control units in separate network messages that are not conflicting with the communication of the base functionality.
- *Download software through a trusted party*. The system should not allow any other system to download apps into it, but rather the ES itself should initiate the download from a trusted

party, which could for instance be the owner's or manufacturer's server. Other systems can thus only send a request to the ES that it should initiate the download of an app.

However, there still remains an area of fraud not so common in traditional ES, namely that other devices send erroneous information with the purpose of deceiving. One could easily imagine situations where considerable damage could be caused by such acts of sabotage. Consider for instance vehicle-to-vehicle communication, where a vehicle sends out a warning to the vehicles behind it that it is braking heavily. A saboteur could construct a piece of software that externally behaves like an automobile on the communication channels, and that suddenly sends out such a warning, claiming to be at an arbitrary position on the road. Placing the saboteur on the side of the road with this device in rush-hour traffic could certainly cause serious traffic disturbances, or even accidents.

It is hard to come up with general mechanisms to guard against such misbehavior, in particular since similar situations could also arise due to a fault in an otherwise benevolent device. One possible mechanism to raise the threshold would be to certify devices, making it harder for the saboteur's device to qualify as a member of the federation, but this is by no means foolproof. Instead, at some point the ES has to make decisions on how reliable the received information is, by comparing it to the information received from other parties and from its own sensors.

To summarize, security questions pose considerable and important challenges to the design of federated embedded systems.

4.2 Interoperability

In this section, interoperability issues in federations are discussed, both on a high abstraction level concerning the ability of FCs to understand each other, and shortly on a lower level dealing with communication specifics. Finally, the effects of mobility are discussed.

4.2.1 Interaction standardization

Interoperability was another issue that came up repeatedly during the workshop discussions. Common communication and interaction standards are crucial for the formation of open, flexible federations of heterogeneous ESs that are envisioned in this work. Ideally, it should be possible to compose ESs into federations in a plug and play fashion, which requires not only widely accepted standards but also robust middleware to simplify the interactions.

Today, there exists a number of standards for machine-to-machine communication, such as the KNX standard for home and building control, DSRC (Dedicated Short Range Communication) for vehicle communication, StanForD (Standard for Forestry Data and Communication) for communication between forestry equipment, DLNA (Digital Living Network Alliance) for enabling home electronics communication with computers and mobile phones, CIM (Common Information Model) for information exchange between application software in electrical networks, IPSO Alliance standard for smart objects based on the Internet Protocol, ZigBee Alliance for wireless sensor networks, and many others. However, generally these standards are domain specific, which makes them less useful for those federations that cut across several domains. Also, the standards are quite specific and undergo frequent changes, making it hard to keep up with them and making interconnected products obsolete rather rapidly.

An attempt to provide a general, domain independent standard is made by FIPA, the Foundation for Intelligent Physical Agents. FIPA's focus is on communication between agent-based systems, i.e. systems that do not simply react to external requests, but have a say of their own about whether to heed these requests or not. Agent-based systems are sometimes considered with certain skepticism, mostly due to the unpredictability about the agent behavior that often results from the inbuilt agent intelligence. However, in its simplest form, an agent is strongly related to an FC of our setting, the latter being ultimately responsible for its own safety and profit, with the possibility to refuse heeding unbeneficial requests from the federation.

The FIPA standard seems well positioned to serve as a basis to provide interoperability in federation applications, in practice probably extended with some domain specific attributes. The challenge here is to apply the best elements of the FIPA, or some other, standard to real applications in order to gain more knowledge about its drawbacks and benefits. Also, it is important that the communication standard does not sacrifice any of the mentioned system properties, such as safety, security, predictability, etc.

4.2.2 Communication

The low-level specifics of communication technology, such as coverage, energy consumption, transmission protocols, etc. were out of scope of the workshops and were thus not explicitly addressed. However, some aspects of such technology came up naturally during discussions.

One such aspect that affects the design of application functionality is whether to opt for 3G, WiFi, or wired communication. WiFi only allows short range communication, leading to the need of more routers, while 3G is less reliable and potentially more costly, especially if roaming is activated. Communication shadows, leading to absent or oscillating signals, should also be taken into account. Communication delays may also play an important role in some applications.

4.2.3 Mobility (both physical and logical)

Mobility, both physical and logical, i.e. ESs joining and leaving a federation, has a direct impact on the communication within the federation and in consequence on the emergent functionality. Well defined mechanisms for finding and joining federations are needed, with the time requirements on them becoming tighter as the mobility level increases.

Conversely, signal losses due to mobility should be accounted for by the federation designer, whether they are caused by an ES moving into an area with known coverage problems or by the more unpredictable situation of an ES suddenly leaving the federation. This is especially important in cases when the disappearing ES emits or retransmits unique information. Adaptation of FC behavior and re-distribution of responsibilities, in response to changes either in the federation composition or in the surrounding environment, are important questions to consider.

4.3 Data management

Information flows are central to the idea of ES federations, having an impact on several aspects of federation design. This raises a number of interesting questions.

One safety related challenge is how to separate data into different classes of criticality or privacy. Another open question is how to store data so that efficient information exchange is enabled, keeping in mind that some applications will need to access data in real time. Of course, this question is further complicated by the fact that the amount of data that will normally be shared in a

federation is expected to be quite high. Also, the same or similar information may often be collected, and possibly stored, in different FCs. This opens up for combining the information from different sources in an intelligent way, thus creating a better picture of the environment that an individual FC could do on its own. But on the downside, this leads to an even higher data load.

The question of assessing the reliability of received data, with direct consequences on the actions of the receiver, was already mentioned. Such assessment can depend on a number of factors, such as the trust value of the sender which may be based either on its previous action or on its technical characteristics (such as the precision of measurements), the number of routers that were involved in transmitting the signal, the location of the sender, the time of signal emission, etc. Further, the assessment could be done in each receiving FC or in a dedicated FC acting as a trust assessment mediator.

4.4 Architectural design

Any federation is organized, either explicitly or implicitly, into a number of structures, defining how such concepts as data storage, communication, control, etc. are distributed. In certain cases, possible organizations will be restricted by either the available technology or the requirements of the individual FCs or their stakeholders, while in other cases the federation designer will have more freedom.

Conscious federation design will require a better understanding of the drawbacks and benefits of different organizational structures and how they affect the emergent behavior and characteristics of federations. As an example, the distribution of control includes such questions as where to gather which data, where to best perform computations, which actuators should be effectuated, etc.

Taking it one step further, general architectural principles for systems of systems should be defined, aimed at promoting desired emergent behavior in federations. Some important federation characteristics are robustness, predictability, flexibility, quality of service, re-configurability, scalability, privacy, safety, and security.

4.5 Heterogeneity of components

The heterogeneity of components is assumed to be high in the envisioned federations, due to cross-domain applications, many competitive producers, different versions of the same hardware or software, etc. Furthermore, there will be ESs, sometimes with unknown characteristics, affecting FCs without participating in the federation.

Also, each ES will potentially participate in several federations simultaneously, leading to different combinations of active apps in each FC, increasing both the overall heterogeneity in the federations and FC internal software complexity. Ensuring safety and predictability in the face of heterogeneity of components is a formidable challenge.

4.6 Life-cycle management

Naturally, life cycle management of federated ESs will be more complex than is the case for traditional ESs. For example, basic software (or hardware) upgrade may lead to incompatibility with the already installed apps. Incompatibility may also arise after app upgrade due to unpredicted interactions between the apps or between apps and basic software.

How to configure FCs in a comprehensible manner, allowing non-technicians to do it, is another challenge, even more so if newly joined FCs may affect the configurations of the other federation members. Also, configuration data and other stored data, such as for example trust values for other FCs, may need to be maintained throughout an upgrade.

The questions of certification of either app developers or apps must also be solved. Priority of apps or ES actions may need to be pre-defined for the case of conflicting requests. There should be secure authorization mechanisms for app installation and uninstallation. Taking this argument one step further, the ownership of each aspect of the federation functionality should be clearly assigned to a stakeholder.

The openness of federated ESs towards third party apps and the extensive information exchange within federations is likely to have a significant impact on the development processes. Thus, deeper investigation of necessary methodology and tools will be needed.

In order for the ESs to be willing to participate in federations, attractive business models will be necessary. Generally, federations will be formed on an opportunistic basis. Thus, the question of how to influence such opportunistic process merits further research.

Legal issues need to be considered in certain applications. For example, prototype technology for platooning applications has been available for some years. However, it is still not near reaching the market due to the common mistrust in technology, especially when it comes to safety critical applications, such as platooning. Questions of legal responsibility in case of failures become even more complex when there are many stakeholders involved in a system, as is the case in a typical federation application. This reasoning can be generalized into considering other factors that affect the human acceptance of the new technology.

4.7 Efficient coordination

Efficient planning of interactions within a federation is imperative. As mentioned previously, such planning can take the form of scheduling, feedback control, or negotiation. For each of these cases, efficient algorithms that scale with the number of FCs are needed.

Also, it is important that the planning algorithms are prepared for a dose of uncertainty. In fact, federations can seldom be fully predictable, due to changing environments, a certain degree of autonomy that each FC enjoys, and transmission challenges, such as noise, latencies, etc. As a complement to planning, supervision of the emergent functionality should be done in order to make sure that the coordination algorithms actually work acceptably and are heeded by the FCs.

Further, it would be interesting to study the balance between federation global and FC local planning. This includes the already mentioned balance between SoS level and FC level availability, taking it one step further by also considering how the quality of service and FC's meta-functionality is affected by the different actions within a federation. Conflicts are likely to occur, both between FCs and between local and global plans. Thus, further work on conflict management is needed.

In particular, simulation environments are of great importance to understand and predict the emergent behavior in federation applications, as well as its relation to FC level functionalities.

5 Conclusions

This report investigates the vision of future embedded systems that no longer act in isolation, but are increasingly communicating with other systems, which creates possibilities to offer new and better services within many technological domains and even across the domains. It is believed that this type of systems of systems, sometimes referred to as internet of things or cyber-physical systems, will be the next large technological leap, comparable to the emergence of internet [31].

In this work, the concept of federated embedded systems was discussed from several perspectives during a series of workshops with participants from both industry and academy. The goal was to lay a foundation for the future research and development work within this area by highlighting common characteristics of such systems and research challenges that they entail.

First of all, a portfolio of applications examples, centered on interacting embedded systems that cooperate in federations to provide new services, was collected, see Chapter 2. It was shown that the idea of federated embedded systems is applicable to a vast number of domains, such as automotive, energy, manufacturing, agriculture, transportation, and healthcare. Also, it became clear that the concept of ES federations is gaining ground rapidly in new products and services that are starting to reach the market.

From the research point of view, it is important to keep up with the evolution of the market, providing efficient methods, tools, and processes for the development of federated ESs, allowing them to be robust, resilient, dependable, heterogeneous, mobile, and large-scale. It is a significant challenge and requires at the very least a good understanding about what traits such systems may have in common.

On the basis of the application examples, early conclusions about common characteristics in ES federations were drawn, see Chapter 3. A conceptual model was proposed, including both technical and non-technical concepts, such as embedded systems, apps, federations, rules, protocols, producers, federation designers, emergent functionality, and others. Next, different life-cycle considerations that are affected by the possibility to flexibly join and leave an ES federation were examined. Also, different software ecosystem models were discussed, inspired by current trends in ecosystem design. Further, it became obvious that most federation characteristics as well as the development processes are strongly dependent on how different aspects of federation functionality are organized. Thus, the topic of organizational structures received special attention. Finally, a reference architecture, containing common functionality to most application examples, was proposed. The reference architecture was structured into three parts, related to planning, supervision, and data management functionalities.

The common characteristics of federated ESs crystallized quite naturally into a set of challenges, both technical and non-technical, see Chapter 4. Safety, security, and reliability were issues that were highly ranked by the workshop participants. Related issues of considerable importance are privacy and maintainability. For example, how to update both basic and application software in federated ESs without compromising safety is not a simple question. Efficient planning algorithms and negotiation protocols are also central to the future ES federations. It is conceivable that planning will be carried out by different actors and at different federation levels. This may lead to conflicting control decisions, making conflict and fault management important challenges.

When it comes to the less technically oriented aspects, the most prominent challenge, mentioned repeatedly during the workshops, was the need for common, cross-domain standards. Further, in order for the federations to evolve as desired, appropriate business and software ecosystem models will be needed. In some applications, especially those containing elements of safety criticality, the distribution of responsibility, sometimes including legal considerations, should be considered before deployment. In summary, different tools, models, and processes for developing ESs that are able to safely participate in different kinds of federations and possibly uncertain environments will be needed.

As mentioned previously, this pre-study is meant to serve as a foundation for future research within the area of federated embedded systems. The next natural step is to choose a couple of application scenarios, guided by the actual industrial needs and visions, and use them to address more profoundly those challenges that were the most significant to the workshop participants, including questions of safety, predictability, interoperability, data management, etc.

6 Appendices

6.1 Appendix A – Application analysis

In this section, characteristics and challenges of some of the applications presented in Chapter 2 are discussed using the concepts introduced in Chapter 3. The level of detail in the following analysis depends on how much attention the application received during the workshops. Some applications are discussed at length, while others are covered more schematically, using the notation from Chapter 3, and in particular the conceptual model and the reference architecture presented there. The analysis of yet other application examples is left out of this report since they were not discussed during the workshops.

6.1.1 Route planning

6.1.1.1 Conceptual model

- ESs: vehicles (required), possibly with internal route planning app.
- GPCSs: central route planner (optional), auxiliary roadside infrastructure (optional).
- Environment: surrounding vehicles.
- Apps:
  - An app in each vehicle that transmits information about the speed, position, and destination of that vehicle.
  - A planner app that aggregates information about the traffic situation and proposes the best route for each federated vehicle.
  - Optionally, an app in roadside infrastructure that facilitates information passing between vehicles and planners.
- Functions: normal vehicle functions
- Users: drivers of the vehicles
- Producers: vehicle manufacturers; app developers.
- Owners: vehicle owners; infrastructure (incl. planner GPCS) owners; app owner (e.g. the road transport administration).
- Communication channel: wireless, preferably WiFi 802.15p
- Beneficiaries: society; drivers.
- Emergent functions: route planning.
- Federation designer: app developer or app owner.
- Federation supervisor: app owner.
- Roles: vehicles; planners (may coincide with the vehicle role); routers (optional)
- Rules (vehicles):
  - Vehicles should share information about their position, speed, and destination with the planner.
  - If the planning functionality is distributed to the vehicles, they should share information about their position and speed with each other.
- Rules (planner):
  - Planners should be able to receive position, speed, and destination information, aggregate the data and return route suggestions in real time.

6.1.1.2 Reference architecture

The planning functionality depends on the choice of organizational structure. The planning itself seems to be a suitable task for some scheduling algorithm. However, if the planning is distributed, different vehicles may opt for different, and sometimes conflicting, schemas with the additional need for negotiation protocols between them.

Fault handling is probably best distributed to each FC, while the supervision functionality could be co-located with the planning. For example, anomalous speed information could be detected by letting the planner compare data from nearby vehicles. The planner should also be vigilant towards malicious attacks with the goal of creating chaotic plans that do not reflect the traffic situation.

Data analysis, and thus also data storage, is probably only needed at the planner side, while some data aggregation may be done by the individual vehicles and possibly by some intermediary roadside infrastructure.

6.1.1.3 Challenges

- Privacy should be considered, especially if vehicles exchange information directly with each other.
- How to delimit the geographical zones within which the information is shared?
- Efficient planning algorithms are needed.
- In case of distributed planning, well-defined negotiation protocols are needed.
- This application is not safety critical, but probably quite susceptible to malicious attacks.

6.1.2 Smart traffic lights

6.1.2.1 Conceptual model

- ESs: vehicles, traffic lights.
- GPCSs: central planner (optional).
- Environment: surrounding vehicles, road infrastructure, pedestrians.
- Apps:
  - An app in each vehicle that transmits information about the speed, position, and possibly destination of that vehicle. It also receives a reference value for the velocity and directs it to the appropriate control mechanisms within the vehicle, such as the cruise control.
  - An app in each traffic light that transmits planned green slot periods and receives recommended adjustments to such slots.
  - A planner app that aggregates the traffic information in the vicinity of an intersection and transmits the reference speed to the approaching vehicles, aiming at reducing the total amount of starts and stops. Such a planner can reside either in a central GPCS, in the traffic lights, or in the vehicles themselves.
- Functions: normal vehicle functions, incl. some sort of cruise control.
- Users: drivers of the vehicles
- Producers: vehicle manufacturers; app developers.
- Owners: vehicle owners; infrastructure (incl. planner GPCS) owners; app owner (e.g. the road transport administration).
- Communication channel: wireless, preferably WiFi 802.15p
- Beneficiaries: society; drivers.

- Emergent functions: more balanced traffic flows through intersections with a reduced environmental impact.
- Federation designer: app developer or app owner.
- Federation supervisor: app owner.
- Roles: vehicles; traffic lights; planners.
- Rules (vehicles):
  - Send the information on a regular basis about its position, speed, and possibly destination to the planner;
  - Listen for the velocity recommendations from the planner and use them as reference values in the internal velocity control loops;
- Rules (traffic lights):
  - Send the information about the time for planned green slots to the planner;
  - Send the information about the mean speed of the passing traffic to the planner (optional);
  - Listen for the information about recommended green slot times from the planner and adjust these slots accordingly;
- Rules (planner):
  - Listen for the information about the times for planned green slots of the traffic lights
  - Listen for the information about vehicle speeds, positions, and possibly directions;
  - Calculate the optimal green light periods (with respect to the overall traffic flow) and transmit this information to the concerned traffic lights;
  - Plan the velocities of connected vehicles and transmit these plans to the vehicles.

6.1.2.2 Reference architecture

- Planning:
  - Feedback control in the case of centralized planning strategy;
  - Negotiation based planning in the opposite case, when each vehicle and each traffic light has full control over its speed and green slot periods, respectively.
- Planning assistance:
  - Traffic lights may monitor the surrounding traffic and alert the planner if some known traffic patterns are recognized.
  - Participating vehicles may inform the planner about vehicles in their environment that are not connected to the smart traffic lights federation.
  - Vehicles can be assigned trust values that show how well they have followed velocity recommendations earlier (this might have to be dependent on the driver).
- Data analysis:
  - Pattern recognition can be used to make rapid forecasts about future traffic situations.
- Data aggregation: is probably made at all levels and FCs of the federation.
- Data storage:
  - Trust values may need to be stored, probably in the vehicles themselves due to scalability reasons;
  - Known patterns may be stored with the planners or in a central pattern database.
- Supervision:

  - It is assumed that the vehicles perform their own internal supervision to avoid dangerous situations that may arise either due to other FCs not heeding velocity recommendations or due to the presence of non-connected vehicles.
  - Conflicts between different planners need to be detected.
  - Supervision of how well the vehicles heed the velocity recommendations is needed in order to update their trust values. Such supervision could be placed locally in each vehicle app (even though the risk for manipulation of trust values seems to be rather high in such a case).
- Fault handling:
  - Each vehicle is also responsible for its own fault handling.
  - In case of negotiation based planning, conflicts between vehicles or adjacent traffic light regions need to be resolved on a higher level.

6.1.2.3 Challenges

- Negotiation protocols and conflict handling in case of distributed planning.
- Efficient algorithms for data-intensive computations in case of centralized planning.
- How to find and register with the appropriate planners?
- Communication failures.
- Trust assessment.
- Privacy.

6.1.3 Cooperative pre-crash systems

6.1.3.1 Conceptual model

Federation components are vehicles and possibly some roadside infrastructure. Due to the need to react rapidly, it does not seem probable that there would be enough time to communicate with some external (to the vehicles) central processing unit.

As it looks today, the software that governs this kind of functionality will, due to its safety criticality, be developed be the vehicle manufacturers themselves and deployed with the vehicle (no apps). If roadside infrastructure makes part of the federation, some sort of certification by the legislative authorities will probably be required for its producer.

Communication is wireless, in a mobile, constantly changing, environment. Currently, the automotive industry is working on a new Wireless LAN standard for communication, WLAN 802.11p, with a dedicated frequency (at least in Europe) for car-to-car communication. Since the communication range of this technology is only a few hundred meters, vehicles should be able to serve as routers and forward messages. Such forwarding requirement seems necessary, but seems to contain potential risks of communication congestion. If so, this might be a reason for using roadside infrastructure to spread information in an efficient manner.

The emergent behavior of the federation is to detect pre-crash situations through communication with surrounding vehicles and deploy necessary safety measures. This is highly safety critical and will have precedence over most, not to say all, functions in the individual vehicles. Special attention at avoiding false alarms is thus recommended.

The design of the federation functionality on a high abstraction level, for example what messages to pass between the vehicles in case of emergency and how to do it, is achieved through common efforts of the automotive industry. On the lower level, each vehicle manufacturer will probably have quite some freedom to implement the required high-level functionality.

Supervision of the functionality could be left to a certification or a testing body, for example in a similar way as is done today with car crash testing (e.g. by Euro NCAP).

6.1.3.2 Reference architecture

Due to safety and time criticality of this application, all planning will probably be done locally in each vehicle. Roadside infrastructure may act as a planning assistant by forwarding data to the vehicles that are farther away from the place of imminent accident. Since timing becomes less critical the farther away one gets, the roadside infrastructure could process its data before transmission, for example to recommend a gradual velocity decrease in function of the distance to the accident.

Forecasting is an important ingredient of planning in this case. After all, the whole point of the application is to correctly predict a crash and to act accordingly. This may be the most critical part of the functionality, both with respect to passenger safety and vehicle brand reputation.

Since planning is distributed (and must be done rapidly), each vehicle will probably keep its own data (data patterns and environment models), allowing it to make correctly detect critical situations. Supervision of the functionality and fault handling is also left to the individual vehicles. Using other words this is called diagnosis and fault handling and is a natural part of the functionality in modern vehicles.

Conflicts should mostly be avoided by design, since the time for handling them will most often be too short. However, an important question that should be addressed is when to trust the information from other federation components above the data from a vehicle's own sensors.

6.1.3.3 Challenges

- Many software producers (as many as vehicle manufacturers), which calls urgently for standardization. This is ongoing work within the automotive industry.
- Wireless communication in mobile, and thus constantly changing, environment, which always contains the risk of communication failures.
- There might be a risk for congestion in the communication network during rush hours due to the requirement of being able to forward messages between the vehicles.
- The safety criticality of this application and its far reaching consequences for the vehicle behavior place an extremely high importance on correct detection of critical situations. An undetected crash is clearly a bad thing, and even more so if the vehicle manufacturer has made some safety promises. False alarms, on the other hand, may lead to unnecessary crashes due to safety braking, with an even more difficult situation for the vehicle producer.
- Conflict handling should be reduced to a minimum. However, the line between when to listen to other FCs and when to decide on its own should be clearly defined, not least for liability questions.

6.1.4 Intersection collision avoidance

6.1.4.1 Conceptual model

Vehicles and roadside equipment are natural federation components, communicating with each other wirelessly. Normally, only the vehicles nearby an intersection are of interest. Thus, WLAN technology (and specifically the new automotive standard WLAN 802.11p) could be used.

Vehicles need to understand warnings coming from the infrastructure and either present this information to the driver or use it to adapt their speed. A return handshake message should be sent from the vehicles to indicate that the warning has been received. Such return receipt could also include information about the action that the vehicle is about to take.

The environment is mobile and thus prone to communication failures. Federation enabling software will often be developed by the vehicle manufacturers themselves (seems to be the normal state of affairs since this application is included into the vision of the future agreed upon by the automotive industry), but since it probably does not belong to the highest safety class, it could as well be developed by a 3rd part. The federation rules however will most certainly be designed in common by the automotive industry.

The emergent function may interfere with the individual vehicle function, for example if a request to slow down comes while the driver is not in the mood to run slowly. Of course, the inconvenience from such interference should be smaller than the one that comes from normal traffic lights (which is quite accepted by the general public nowadays). The other direction of affection, i.e. when an internal vehicle function (or dysfunction) affects the emergent behavior, should be detected by the federation enabling software. Ideally, the other federation components should be notified about it.

Supervision of functionality and ownership of the roadside equipment could be delegated to the traffic administration authorities.

6.1.4.2 Reference architecture

Planning will be probably done in a centralized manner, either directly by the roadside equipment at each intersection or forwarded to a central service. The type of planning is scheduling. Ideally, each vehicle in the federation should assist the planning procedure by predicting its proper velocity and/or time of arrival at the intersection, as well as by transmitting the information about surrounding non-federated vehicles that it might detect with its radar, WLAN, or other equipment.

Some sort of fault handling will probably be present in all federation components. Even if there is always a driver to fall back on, certain information may be useful to alert the driver, for example the information about malfunctioning roadside equipment. As a special case of fault handling, conflict handling should be previewed by the federation designer. Clear rules are needed for how to handle simultaneous vehicle arrivals from different directions.

Also, it is not too difficult to imagine that some drivers may try to manipulate the system by mixing with their federation component in order to send deceiving information (to get free way through the intersection). This could for example be dealt with by using the information from other vehicles that is available in the system. One could also punish the deceptive vehicle by reducing its trust value (in some way).

The need for data storage seems to be relatively low. Of course, the planner needs to keep some data about the intersection environment during its planning procedure, both static (such as connecting roads) and dynamic (such as approaching vehicles, traffic jam at next intersection, etc.). If the federation components warn each other about unfederated vehicles, the fact of sending a warning message should preferably also be remembered for a while to reduce the communication load. The ID's of untrusted vehicles could be saved in a planner. If planning is distributed to the roadside equipment, it might be worth to share this information with other planners.

The reverse trust model, showing the level of trust to the planner, could also be useful. It could be stored either in the vehicles or in the planner itself (and communicated to the vehicles). For example, if the traffic is intense, the planner may know that is not capable of sending warnings to all vehicles. If this information is somehow communicated to the vehicles, they may notify the driver about the need to be more alert.

6.1.4.3 Challenges

- Standards are needed for the vehicles to understand the warning signals from the infrastructure. An alternative is to let a 3rd part develop the federation enabling software.
- Possible communication failures due to mobility.
- There is a good ground for conflicts between the vehicles approaching an intersection from different directions.
- There are clear advantages for the vehicles to send deceptive signals, which means that sooner or later it will occur. Thus, trust models or some other way of handling deception is needed.

6.1.5 Cooperative driving / platooning

6.1.5.1 Conceptual model

Vehicles that have the equipment to participate in a platoon are natural federation components. There may also be some roadside infrastructure to aid the inter-vehicle communication, as well as GPCSs to process data and aid in decisions about platoon formation and dissolution.

Natural roles are those of a leader and a follower. It may be advantageous though to split the follower role into a normal follower, containing all vehicles except the first and the last, and the last follower, consisting of the last vehicle in the platoon. If the platoon scheduling is done centrally, a planner role is needed. A router role may be needed for the quality of service, either played by the vehicles themselves or by some road-side infrastructure. To reduce the communication load, all platoon external communication, e.g. communication with other platoons, a scheduler, or a candidate vehicle, could be handled by one vehicle playing the role of a representative.

In the case of privately owned vehicles, its user will often be the same as the owner, while the situation is normally the opposite when it comes to the commercial traffic. However, in that case drivers are often personally linked (for example through salary levels) to the interests of the owner, resulting in a similar desire as the private drivers to participate in a profitable platoon.

As the situation is developing at the moment, it looks as if each vehicle manufacturer will strive, at least to begin with, to come up with a proprietary version of the application software. Standardization efforts are obviously needed, similarly to the development of the IEEE 802.11p

communication standard. The multitude of producers, working with the platoon idea, also makes it difficult to easily assign the roles of federation designer and federation supervisor.

The emergent functionality is a better traffic flow, leading to societal and personal benefits, and a smaller travel cost, due to less wind resistance. The price for this is that a vehicle will need to adapt its velocity to the platoon, which may be in a conflict with the driver's preferences. Of course, if the conflict is large enough, the solution is to simply choose another platoon. However, if the conflict is small, or if the driver's moods change from day to day, the right decision is harder to take. The best compromise is probably to let the driver decide in the limit cases. The challenge is then typical to most pervasive systems, i.e. how to know for the app about when to notify the driver and how to present necessary information in a simple way, without drawing too much of the driver's attention. An important requirement that each individual vehicle must meet, besides avoiding to annoy the driver, is to always monitor and react to situations that may affect the vehicle safety.

Finally, by the nature of the application, the communication will be wireless, probably using several available technologies. The already mentioned IEEE 802.11p standard is promising for short-range communication, while 3G will probably be used for inter-platoon communication (e.g. for scheduling purposes). Due to the use of 3G, roaming costs should be considered. On the other hand, the importance of 3G could be reduced by making use of the available vehicles and/or road-side infrastructure as routers.

6.1.5.2 Reference architecture

Communication and planning are fundamental to this application. Planning of platoon formations could be done centrally. However, the scale of the problem seems quite demanding for such an approach. This is even further so if the driver preferences (and moods) are included into the planning procedure. It seems more plausible (and scalable) if the decision making about which platoon to join is distributed to the individual vehicles. This calls for efficient communication protocols so that necessary information about the characteristics and requirements of platoons that may be of interest is received. Simple enough negotiation schemes for joining a platoon would then also be needed.

The environment model of each vehicle will contain data from its surroundings, to a large extent dominated by the information about the other vehicles within the platoon. Exceptions to this domination are the leader and the last follower that will have a larger part of their information coming from the world outside of the platoon. It is important that these vehicles detect obstacles and non-federated vehicles. The representative vehicle will also have a broader view of the environment, keeping track of and communicating with nearby platoons and individual platoon-enabled vehicles.

It is imaginable that some FCs will try to gain individual advantage by lying about their arrival times. If the arrival of new FCs may affect the platoon behavior, some sort of trust model / information validation may be needed.

Fault handling is of paramount importance due to the safety criticality of this application. It should be fast, robust, and reliable. Since vehicle manufacturers would be held responsible in case of a system failure, it is in their own interest to develop fall-back mechanisms that prevents accidents. In other words, such mechanisms will most probably be developed in parallel by each automotive

manufacturer. In fact, some of these mechanisms exist already. For example, adaptive cruise control systems use radars and video cameras to maintain a minimal distance to the vehicles ahead.

Due to a large number of possible participants, conflicts are probable and conflict handling should be explicitly considered. It might be worth to partition conflicts into several categories. One obvious partition is to separate safety critical conflicts that may lead to accidents from those that only annoy the driver or lead to a longer travel time. The safety critical conflicts may need central coordination authority or crystal-clear resolution rules, while the non-critical ones may be solved by adaptive algorithms and negotiation.

Each vehicle is supposed to supervise both itself and the state of its environment, to be able to react in time to faults and conflicts. Protocols for transmitting alarm information both within and, through a representative, outside the platoon are necessary.

Assuming that the decision making is left to the individual vehicles, it seems natural to also distribute the data storage. However, some auxiliary services, such as weather forecasts, traffic statistics, etc., could be stored at a central server (GPCS).

6.1.5.3 Challenges

- The safety-criticality of these systems makes fallback mechanisms (ideally, graceful degradation) extremely important, not least to resolve legal issues related to the popular acceptance of the technology.
- To join a platoon and to maintain its position within a platoon, each vehicle will communicate its position and direction. This immediately raises questions about privacy.
- Possible communication failures due to a mobile environment with many FCs close together (interference).
- Strongly dynamic federations. How should this be dealt with, for example when establishing connections, closing connections, living with varying number of participants, etc.?
- What business models would be applicable?
- Some vehicles will not be platoon enabled (white spots), introducing uncertainty about the environment.
- False information attacks will occur to try to maximize individual profit. Thus, some sort of trust handling may be needed.
- How to present information to the driver and how to take its preferences into account?
- Much standardization has been done, but even more is needed.
- Interesting scheduling problem, with many conflicting objectives and immediate intentions to consider.

6.1.6 Emergency lane clearing

6.1.6.1 Conceptual model

- ESs: vehicles; emergency vehicles.
- GPCSs: central route planner (optional), auxiliary roadside infrastructure (optional).
- Environment: surrounding vehicles.
- Apps:
  - An app in each vehicle that receives the information about an approaching emergency vehicle.

  - An app in the emergency vehicle, transmitting the information about its speed and position to the nearby vehicles if the emergency mode is active.
  - Possibly, a planner app that schedules both the emergency vehicle and the vehicles along its route in order to provide as free passage as possible for the emergency vehicle. The route planning functionality could also be achieved by letting the vehicles participate in both emergency lane clearing and route planning federations.
  - Optionally, a router app in roadside infrastructure.
- Functions: normal vehicle functions
- Users: drivers of the vehicles
- Producers: vehicle manufacturers; app developers.
- Owners: vehicle owners; infrastructure (incl. planner GPCS) owners; app owner (e.g. the medical service provider or the road transport administration).
- Communication channel: wireless, preferably WiFi 802.15p
- Beneficiaries: society; drivers.
- Emergent functions: a rapid highway for emergency vehicles.
- Federation designer: app developer or app owner.
- Federation supervisor: app owner.
- Roles: ordinary vehicles; emergency vehicles; planners (optional); routers (optional)
- Rules (ordinary vehicles):
  - Ordinary vehicles should listen for information about the emergency vehicles (either their position and speed or the expected time of encounter) if they are in their active mode.
- Rules (emergency vehicles):
  - Active emergency vehicles should emit necessary information about their planned route to the vehicles in its vicinity, either directly or via some sort of infrastructure.
- Rules (planners):
  - Planners should be able to receive position, speed, and destination of all types of vehicles, and plan their routes so that the emergency vehicles are prioritized.

6.1.6.2 Reference architecture

Since this application is the extension of route planning, with the addition of prioritized vehicles, the reference architecture will be quite similar to the route planning example. The main differences affect the planning algorithm that needs to take different priorities into account. Either a centralized vehicle scheduling or priority based negotiation policy is needed in order to make sure that the emergency vehicles get a free route. It may be important to have a supervision mechanism in place that encourages or even enforces to some extent the ordinary vehicles to accept the priority of the emergency vehicles.

6.1.6.3 Challenges

- Privacy should be considered, especially if vehicles exchange information directly with each other.
- How to delimit the geographical zones within which the information is shared?
- Efficient planning algorithms that can handle vehicles with different priorities are needed.
- In case of distributed planning, well-defined negotiation protocols are needed.

- Supervision mechanisms to ensure that the emergency vehicles actually get the free route are needed.

6.1.7 Hazardous road conditions notification

6.1.7.1 Conceptual model

This application is open for a diversity of different federation components. Four roles are readily identified, senders, recipients, central data processors, and routers of the road condition information.

Most obvious recipients are vehicles, but it could also be smart phones (allowing to make route choices at the breakfast table), radio stations, etc. Weather information could be used by nearby smart houses for their thermostat control.

A sender could be any embedded system that is federation enabled (most often this means that the federation app has been installed on the ES) and equipped with appropriate sensors. Again, the router role could be played by most ES with a sufficient communication capacity. As mentioned in the application description, construction machines or roadside equipment could be good choices to spread the information about traffic conditions.

High-capacity GPCSs are probably the most suitable as central data processors. However, it is imaginable to distribute the data storage as well as the data processing between the other federation components (ES). In such a case, much smarter planning algorithms are needed. Also, it is important to make sure that the data processing does not interfere with the safety critical functionality of the ES. Also, some rules about how it may interfere with non-safety critical functionality would be necessary.

In case of pre-defined actions, triggered by the central data processor, business aspects of how these actions (or services) are handled should be considered. For example, in the case of automatic snowplowing, some natural questions are which snowplowing companies should be alerted, how should they be paid, should the city authorities acknowledge the alert before it is sent out, what if there is a false alarm, etc.?

The emergent functionality of this application is the spreading of information. Normally, it is not safety critical and does not have a direct impact on of the functionality of the federation components. Implicitly however, it might affect the communication capacity and processing characteristics. Also, since playing around with this application would not endanger people but may lead to unpleasant traffic chaos, this might be a quite attractive application for the hackers. Finally, since each road condition notification reveals the position of the sender, privacy considerations are important.

Since this application in fact conceals various applications, depending on what purposes the information is used for, the design and supervision of the federation should be done openly, so that new information sharing application are easily built on top of the existing information sharing infrastructure. Perhaps, this kind of applications may be suited to be developed and maintained by open-source communities, such as Wikipedia or Cosm.com (previously Pachube) [26].

6.1.7.2 Reference architecture

There is no actual need for planning in this application scenario. The closest we get to the coordination functionality is perhaps the information routing. Why and how to motivate FCs to voluntarily route messages to the rest of the federation is an important design question though.

Data is aggregated by each sender FC. Whether it should be analyzed at the sending or the receiving side (or both) is an interesting design question. The location of historical data, environment models, and fault handling will be strongly dependent on the choice of place for data analysis. All in all, this seems to be a rather non-critical, but useful set of applications, with a lot of freedom for the designer.

6.1.7.3 Challenges

- Privacy issues should be considered.
- Not safety critical, but probably attractive to hacker attacks.
- The emergent functionality may affect communication and processing capacity of the federation components.
- No actual planning functionality in this application, possibly except for pre-defined alarm triggers.
- Suitable for open-source development?

6.1.8 Home and energy automation

A combination of smart appliances and smart grid applications was discussed in detail during one of the workshops. In that discussion, the whole electricity distribution chain was considered, from the electricity provider, through smart passive buildings that contained a number of smart apartments, local energy generators and batteries, to the end electricity consumers, consisting of interconnected smart appliances, Figure A-1.

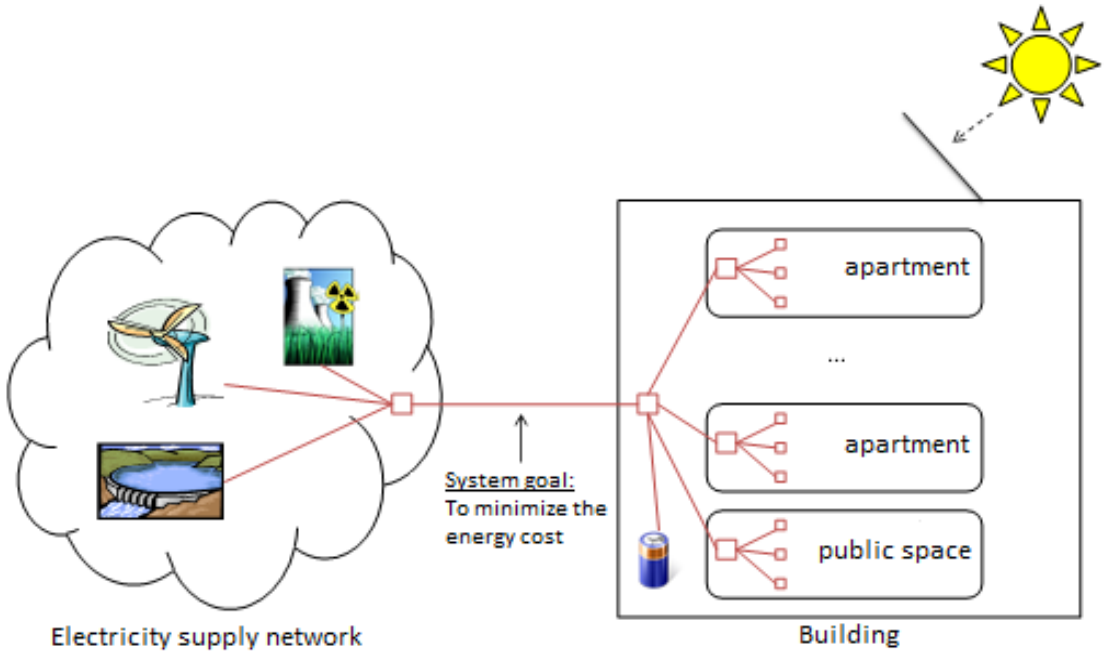


Figure A-1. The electricity distribution chain, from the provider to the smart apartments.

6.1.8.1 Conceptual model

Four federation roles were identified, one for each level of the electricity distribution chain:

- Electricity provider
  - Sets the electricity price
  - Provides price prognoses
- Smart building
  - Controls the energy balance of the building by buying and storing, or selling the locally produced or stored energy, when appropriate. It is assumed here that the legislation has caught up so that a temporary surplus of electricity can be sold to the power grid.
  - Acts as a gateway, with respect to the distribution of both electricity and information within the building.
  - Sets local spot prices (based on the energy balance and current/predicted electricity prices).
- Smart apartment (incl. public spaces)
  - Communicates with the smart building.
  - Provides an interface for the configuration of the appliances within the apartment.
  - Provides visualization of the energy consumption.
  - Keeps track of the apartment's state, for example if there is someone inside the apartment, and acts accordingly.
- Electricity consumer (smart appliance)
  - Responsible for the internal control of its functionality, in relation to its internal state and structure, the state of the apartment, electricity price, etc.

Further classification may be meaningful in specific smart house implementations. For example, the role of electricity consumer will often contain a wide variety of products. Some of them may be fully flexible (or smart), in the sense that they can adapt their energy consumption to the information that is provided to them through the smart apartment. This could even mean that they in certain cases release some amount of stored energy back to the system, for example if the energy storage system of the smart building is modeled as a consumer. Other products will not be as flexible. For example, if someone is watching its favorite TV-show, he will probably not be happy with the smart TV suddenly turning off due to the electricity price rising. The level of openness of a consumer towards sharing its energy could also be modeled as a part of its internal state.

Another distinction can be drawn between basic consumers that are common and often required to all apartments in a smart building (such as systems for heating, air-condition, ventilation, hot-water, local electricity production, etc.), and additional consumers that will show a much higher variety between the apartments (such as lamps, household appliances, etc.).

When it comes to the federation components that will take part in smart house federations, the majority of the consumers will be embedded systems. Already, some smarts appliances are starting to reach the market, for example smart refrigerators that come with food management systems being able to alert users about approaching expiration dates or missing groceries. Another example product that is already out there is a smart oven that among other things can propose individual

recipes for each family member, and to configure itself for a chosen recipe, allowing the user to make simple choices via a touch screen or a smartphone.

The environments of the embedded systems will consist of the other ESs that belong to the same smart apartment, the gateway at the smart building level, the people and the furniture that are located in the apartment, the communication media, etc. Apparently, such environments will be rather static, as compared for example to the surroundings of a moving vehicle. Also, to a high extent, these environments will be quite similar to each other, allowing some intelligent information sharing between the ESs.

Both at the apartment and at the building levels, some sort of coordination of the activities on the lower level may be necessary. Thus, some general-purpose computer systems will probably be needed. The same goes for the electricity provider side, that will need to make price forecasts, share price information, and charge the smart buildings (or directly the smart apartments) for the provided services.

Today, the examples of smart appliances are generally aimed towards specific pre-defined goals and only able to communicate with other products of the same producer or possibly smart phones. Software that enables interaction with other products is developed by that same producer and is already in place when the product is purchased. Such approach may continue to be valid if the number of imaginable applications remains low (for example if the main benefit of a smart house federation will be the reduction of the total energy consumption or electricity cost in the apartment). Of course, sooner or later, the user may require that products of different manufacturers communicate, which will lead to an increased diversity of producers within a smart apartment. In the long run, this would probably open the door for 3rd party producers of apps. An interesting question is how this will affect the process of creation and types of future software ecosystems.

When it comes to ownership, there will be a small number of owners. Of course, in one-family houses, there will generally be only one juridical owner, namely the living family. However, more complex situations will also occur frequently, for example when an apartment is rented with furniture from a private person in a smart building, where basic consumers are owned and maintained by the building owner (for example HSB in Sweden). In that case, consumers (and interests) of three owners will have to co-exist.

Assuming that the smart appliances will have to function in different environments and interact with the products of different producers, there will be an urgent need to be able to adapt to a number of different communication media (e.g. wireless, through the electricity net, etc.) and standards (e.g. ZigBee, KNX, etc.). This should be taken into account when designing the products. A flexible and standardized communication language is needed. Such language should be able to express price information, the state (or status), configuration, and API profile of the federation components, possible communication patterns, etc. Also, it should be possible to add new message types as new FCs join the federation.

Also, it is important that the federation components at some point of time are configured to know the boundaries and settings of each federation. Assuming that there is one instance of the smart house federation per apartment, this could be done at the apartment level. How this should be done in practice though, is an open question. Also, it is conceivable that the smart appliances have internal

mechanisms to learn the user preferences and adapt their settings. Some alternative approaches to bind a computer system to a federation were discussed during one of the workshops:

- The main GPCS unit of the apartment could ask for the identities and of all consumers within its WiFi-range. The answer should be returned through the electricity net. In such a way, the apartment GPCS should be able to trace which responses come from within the apartment.
- Another alternative is an NFC-key (Near Field Communication) that is paired with the main GPCS unit of the apartment. It should be sufficient to do this only once, when a new appliance is installed into the apartment.

Returning to the case of using the smart appliances so as to reduce the peaks in electricity price, the beneficiaries are not only the inhabitants of the smart apartments (cheaper electricity), but also the electricity providers (since they need less buffering capacity), and possibly the society as a whole (since less buffering capacity may lead to a smaller environmental impact of the electricity production).

Finally, it is not all too clear how to distribute the responsibility for the supervision and maintenance of the emergent functionality in a smart house federation. Perhaps the simplest would be if the electricity provider could do the supervision, since it might be best positioned to detect abnormal functioning of the federation (or a collection of federation instances) through a sudden increase in electricity demand peaks. Once a malfunctioning is detected, the responsibility for the next step, namely of fixing the system, seems even more problematic, especially in the case of a wide diversity of FC producers. This could be an argument for leaving the design (and the responsibility) of maintaining the federation functionality to a 3rd part (an app developer company).

To summarize, the following conceptual model could be outlined for the home and energy automation application:

- ESs: electricity distribution network; electric vehicles; home appliances; storage capacities; solar panels.
- GPCSs: building energy management system; home energy management system; electricity network management system.
- Environment: Includes calendar data (for example regarding intended use of electric vehicles), other appliances, weather forecasts, etc.
- Apps:
  - An app in each smart appliance that controls its energy consumption.
  - An energy management app in each household.
  - An app placed centrally in the building that represents the apartments and communicates with the electricity network. It also handles common functions in the building (such as water heating).
  - An electricity management app at the global energy provider that dynamically adjusts electricity pricing and actively tries to buy electricity if needed.
  - An app in each electric vehicle, responsible for keeping an appropriate charge level for the user's needs, while storing and selling surplus energy when possible.
- Users: home and vehicle owners; electricity distribution companies.
- Producers: appliance OEMs; vehicle OEMs; energy distribution companies; app developers.

- Owners: home and vehicle owners; electricity distribution companies; building owners.
- Communication channel: wired and wireless.
- Beneficiaries: home and vehicle owners; electricity distribution companies; society.
- Emergent functions: More balanced energy usage, with cheaper electricity bills for the individual users and less backup power equipment needed.
- Federation designer: electricity distribution companies; app developers.
- Federation supervisor: electricity distribution companies; energy management systems.
- Roles: appliances; electric vehicles; home planner; building planner; network planner; storage capacities; solar panels.
- Rules:
  - Building planners and electric vehicles should be able to receive both electricity pricing forecasts and unexpected action requests from the network planner in order to plan the home and vehicle electricity consumption;
  - Storage capacities should communicate their status to the home planner.
  - Home planner communicates with the building planner about predicted electricity prices and expected electricity usage.
  - Smart appliances should supply their technical specifications and let their activity be scheduled by the home planner;
  - Home planner should take into account user preferences.

6.1.8.2 Reference architecture

Every level (or role) of the above smart house application will probably have a planning function. However, at this point it is not clear exactly how the planning responsibility is distributed across the levels. Roughly, it seems plausible that each consumer will control its own internal functionality, while some central apartment planner will coordinate the interactions between the consumers. However, in certain cases direct consumer-to-consumer coordination may occur (in our setting referred to as negotiation), especially between consumers designed by the same producer or in cases where the level of standardization is high (in fact, this is how LG's smart refrigerator and smart oven work today).

On the building level, decisions about whether to store, buy, or sell energy will probably be taken (even though this could also be built into the logic of the individual apartment planners). Also, the smart building acts as a planning assistant to the apartments by forwarding the pricing information from the electricity provider (or even reference values for the energy consumption), collecting weather forecasts allowing to predict future local production capacity, etc.

The electricity provider on its hand should also have a planning functionality in order to set the correct price at each time instant, thus pursuing its own goal of reducing the demand peaks. Ideally, the (central) producer's planner should take into account the state of local producers (smart buildings), which may include their predicted energy production rates, energy demands, and hardware statuses.

Again, the intermediary role of the smart building, being a planner assistant not only to the apartments but also to the central electricity producer, is emphasized. This role may become quite delicate since the assisted parties have somewhat opposed objectives. While both are interested in reducing demand peaks (since it plays in hand with a lower energy cost), during normal operation the

producer will try to maximize its profit, while the apartments will always strive for minimizing their costs. Clear rules for handling this kind of inter-level conflicts should be well-defined by the federation designer for the potential federation components to be willing to participate in the federation.

As discussed above, the consumer environments will be shared to a large extent. Thus a common core environment model could be centrally placed on the apartment level, with local extensions in each consumer. The same goes for the data storage. However, as always, such approach may be efficient but is less robust due to the centralized environment model / data storage being a single point of failure.

Data will be aggregated by the individual consumers, as well as provided from higher federation levels (for example the pricing information will come from the central electricity provider while the smart building may retrieve weather forecasts from the internet). Since the environments are rather static, different consumers will provide measurements of the same physical entities, allowing to derive a more precise model of the shared environment, whether it is done centrally in the apartment planner or locally in each consumer by some sort of model synchronization.

Fault handling was not explicitly discussed for this example application. However, a conclusion from the general discussions on this topic is that every federation component should have a fall-back state in case of unforeseen faults. For the consumer's part this could amount to returning to the non-smart functionality, i.e. the way these products would function today, without interaction with the outer world. Other typical consumer states that come to mind are the low and high energy usage states. The internals of what actions these states lead to and when state transitions should occur are out of scope of this report. However, it should be said that some of this transition and action logic could be defined by a 3rd party app producer.

Often, a consumer will have a better model of the surrounding world thanks to the rest of the federation than it would have if it only could access its own basic functionality. This leads to the interesting question of where to draw the line of responsibility between the safety mechanisms that the app producer defines and those defined by the producer of the original product.

A special case of fault handling occurs when there is a conflict in the requested behavior of different FCs. A simple example of this is how to choose which consumers to run when the electricity price is high. Again, well-defined rules are important. Also, conflicts may occur within a consumer, due to different reference values for the same signal, originated from different apps. Several possible solutions were discussed, all with their advantages and drawbacks:

- Set a priority to each app before its deployment (works only if the number of apps is small).
- Stack apps on top of each other, so that the last installed app must pass through previous apps before reaching consumer's API (this may lead to a large communication overhead).
- A middleware detects conflicts and falls back on the basic functionality if necessary (leads to a predictable behavior, but does not resolve the conflict, which may lead to both conflicting apps being disabled eternally).

6.1.8.3 Challenges

- Planning functionality will probably be needed on all federation levels (both hierarchic and distributed control).
- Design, supervision and maintenance of the federation functionality seem complicated due to a large number of CS producers.
- Well-defined and flexible standards are needed to allow truly open federation (as opposed to federations of products from the same producer).
- The federation should rather seamlessly be able to adapt to varying user preferences.

6.1.9 Demand-driven multi-modal public transportation

6.1.9.1 Conceptual model

Federation components will be the different transportation entities, such as buses, trains, cabs, PRT units, private cars, airplanes, as well as coordination GPCSs (if the federation is coordinated centrally).

It is quite likely that several federation instances will be in use, probably delimited by geographical position of FC types (e.g. a SL bus federation). However, some interaction will also occur between the federation instances. For example, a long-haulage truck could collect traffic information from the federations along its planned path and adapt its driving schedule accordingly. Another example is when there are passengers travelling on a local bus that need to catch a national train connection. The responsibility for communicating with other federations could be handled either by a federation representative or by each FC. In any case, such inter-federation interaction needs to be planned for in advance.

There will often be several producers in a federation. However, if the federations are delimited according to FC types, the number of producers within the same federation will be relatively small, at least when compared with the applications involving private vehicles (often, a transportation company places large orders with one or two producers).

Ownership structures depend on the federation type. In case of a federation being a single bus company, there is naturally only one owner. However, to truly achieve multi-modal transportation coordination, there should probably be a number of owners that will need to be synchronized their communication protocols. In the case of cab or car sharing, the number of owners is of course much higher. The responsibility for the federation design and supervision also depends strongly on how federations are organized. The main beneficiary is the community as a whole.

The emergent function includes adaptation of transportation capacity to varying needs and traffic conditions, as well as the facilitation of finding the most convenient transportation opportunities at a given point of time and space. In some cases, the emergent functionality will be in line with the functionality of the individual FCs. For example, a cab's goal is to transport as many people as possible, which seems as a good goal for a federation of cabs (even if such more optimized transportation may be in conflict with the individual cabs goal of gaining as much money as possible). In other cases, conflicts between the individual FC functions and the emergent functionality may be larger. For example, when delaying a bus to wait for a delayed train, there will be a delicate balance to strike.

6.1.9.2 Reference architecture

Coordination is at the core of this application. How it is best organized is a non-trivial question. Some sort of forecasting mechanisms for near-future transportation patterns seem necessary. Data will be collected, processed, and stored in many different locations in the federation. Inter-federation data sharing is also of interest. This affects questions of communication, security, and coordination.

Taking into account the somewhat stochastic nature of human transportation needs (at least at the individual level), conflicts between different users and federation parts are quite probable. Thus, to gain public acceptation, this application should provide fast and efficient conflict handling and coordination mechanisms. Fault handling on the other hand is not that critical since the application mostly affects the transportation patterns and do not have a direct safety critical impact. However, faults should not occur too often if the application is to be accepted.

Trust models are important, since the functionality is prone to be attacked by self-interested hackers, trying to improve their own transportation experience on expense of the other travelers. Privacy issues are important since the collection of transportation patterns always starts with the location and plans of an individual traveler.

6.1.9.3 Challenges

- Privacy issues should be considered.
- Prone to attacks by self-interested hackers.
- Inter-federation interaction should be considered early.
- Organizational structure is non-trivial.

6.1.10 Precision agriculture

6.1.10.1 Conceptual model

- ESs: autonomous robots, tractors, add-on agricultural equipment, sensor networks.
- GPCSs: farm management system (FMS), weather forecast system (optional).
- Environment: nearby ESs, soil and terrain conditions, weather conditions.
- Apps:
  - An app in each ES that communicates with the FMS;
  - An app in FMS that coordinates the actions of the connected ESs. Some part of the coordination functionality can be left to the ES app.
- Functions: ordinary functions of the individual ESs, e.g. tractors and add-on equipment
- Users: Farmer
- Producers: agricultural equipment OEMs; possibly smaller companies producing robots; app developers
- Owners: farmer; equipment rental companies.
- Communication channel: wireless
- Beneficiaries: farmer; society (for example if the more precise agriculture leads to a decrease in pesticides usage)
- Emergent functions: cooperative farming with a more precise control of agricultural conditions and less soil packing.
- Federation designer: FMS app developer in cooperation with the farmer
- Federation supervisor: farmer or a third party company dedicated to supervision

- Rules :
  - ESs gather information about the surrounding environment, process it somewhat, and send it to FMS in order to make planning possible
  - ESs should have fault detection and fallback mechanisms.
  - ESs follow FMS plans unless their internal fault handling mechanisms decide on another action sequence.
  - FMS is responsible for processing the information that ESs gather and coordinating the ES actions.

6.1.10.2 Reference architecture

- Planning:
  - A central controller (FMS) is responsible for the overall coordination.
  - Each ES will often have its own internal control loops.
- Planning assistance:
  - ESs should report deviations from the overall action plan to the FMS. This includes unexpected obstacles, both stationary and mobile
- Data analysis:
  - Weather forecasts may be useful.
  - Bad growing conditions or plant illness can either be done directly at ESs or in FMS through image processing (pattern recognition).
- Data aggregation: depends on how much control is left over to the FMS.
- Data storage:
  - Known patterns may be stored with the ESs or in a central FMS database.
- Supervision and fault handling:
  - Each ES performs its own internal supervision and fault handling to detect internal faults and avoid unexpected obstacles.
  - Fallback mechanisms should be present in each ES for safety reasons. .
  - FMS may help in supervision and fault detection by aggregating the information from different ESs.
  - It is conceivable to have a cloud-based service for certification, configuration and supervision of both equipment and apps. This might be quite helpful in this setting since it is common that farmers rent parts of their equipment during peak seasons.

6.1.10.3 Challenges

- Heterogeneous equipment that may be in the farmer's possession only during periods of the year (peak seasons).
- There is a risk of unexpected moving obstacles that must be avoided (people and animals)
- Communication network may be poor (unless one is installed specifically with this application in mind).
- Harsh weather conditions.

7 Bibliography

[1] A. Kobetski and J. Axelsson, " Federated Embedded Systems – a review of the literature in related fields," Swedish Institute of Computer Science, Kista, 2012.

[2] "Cooperative Intersection Collision Avoidance Systems (CICAS)," [Online]. Available: http://www.its.dot.gov/cicas/. [Accessed 2012].

[3] "Grand Cooperative Driving Challenge," [Online]. Available: http://www.gcdc.net. [Accessed 2012].

[4] "LG smart appliances," [Online]. Available: http://www.lgnewsroom.com/ces2012/view.php?product_code=95&product_type=95&post_index=1828. [Accessed 2012].

[5] K. C. Sou, J. Weimer, H. Sandberg and K. H. Johansson, "Scheduling Smart Home Appliances Using Mixed Integer Linear Programming," in *IEEE Conference on Decision and Control*, Orlando, Florida, 2011.

[6] R. Green, L. Wang and M. Alam, "The impact of plug-in hybrid electric vehicles on distribution networks: A review and outlook," *Renewable and Sustainable Energy Reviews,* vol. 15, no. 1, pp. pp. 544-553, 2011.

[7] S. Ramchurn, P. Vytelingum, A. Rogers and N. Jennings, "Putting the "Smarts" into the Smart Grid: A Grand Challenge for Artificial Intelligence," *Communications of the ACM,* vol. 55, no. 4, pp. 86-97, 2012.

[8] "EU eHealth research," [Online]. Available: http://ec.europa.eu/information_society/activities/health/index_en.htm. [Accessed 2012].

[9] "Giraff Technologies AB," [Online]. Available: http://giraff.org/. [Accessed 2012].

[10] "Robotdalen," [Online]. Available: http://www.robotdalen.se/. [Accessed 2012].

[11] "Interface Surgical Technologies," [Online]. Available: http://www.intersurgtech.com/. [Accessed 2012].

[12] "Proteus Biomedical," [Online]. Available: http://www.proteusbiomed.com/technology/. [Accessed 2012].

[13] "eTect," [Online]. Available: http://www.etectbio.com/technology.html. [Accessed 2012].

[14] "The Smart Pill Corporation," [Online]. Available: http://www.smartpillcorp.com. [Accessed 2012].

[15] "Medical Device "Plug-and-Play" Interoperability Program," [Online]. Available:

http://www.mdpnp.org/. [Accessed 2012].

[16] "Personal Health Data Standards, ISO/IEEE 11073," [Online]. Available: http://en.wikipedia.org/wiki/ISO/IEEE_11073. [Accessed 2012].

[17] "Fare/Share New York," [Online]. Available: http://faresharenyc.com/ . [Accessed 2012].

[18] "Taxi Share Chicago," [Online]. Available: http://taxisharechicago.com/. [Accessed 2012].

[19] "Citihaiq car sharing," [Online]. Available: http://www.citihaiq.se/. [Accessed 2012].

[20] "PRT Consulting," [Online]. Available: http://www.prtconsulting.com. [Accessed 2012].

[21] "F. Poulsen Engineering," [Online]. Available: http://www.visionweeding.com/. [Accessed 2012].

[22] "Ambient Awareness for Autonomous Agrigultural Vehicles," [Online]. Available: http://www.quad-av.eu/. [Accessed 2012].

[23] "John Deere Machine Sync Technology," [Online]. Available: http://www.deere.com/wps/dcom/en_US/corporate/our_company/news_and_media/press_releases/2011/agriculture/2011aug25_machinesync.page. [Accessed 2012].

[24] "ION Autonomous Snowplow Competition," [Online]. Available: http://www.autosnowplow.com. [Accessed 2012].

[25] "ION Robotic Lawn Mower Competition," [Online]. Available: http://robomow.ion.org/. [Accessed 2012].

[26] "Cosm.com," [Online]. Available: https://cosm.com/. [Accessed 2012].

[27] B. Horling and V. Lesser, "A Survey of Multi-Agent Organizational Paradigms," *The Knowledge Engineering Review,* vol. 19, no. 4, pp. pp. 281-316, 2004.

[28] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing,* vol. 1, no. 1, pp. 11-33, 2004.

[29] N. Falliere, L. O. Murchu and E. Chien, "W32.Stuxnet Dossier, version 1.4," Symantec, 2011.

[30] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *USENIX Conference on Security (SEC'11)*, Berkeley, CA, USA, 2011.

[31] R. Rajkumar, I. Lee, L. Sha and J. Stankovic, "Cyber-Physical Systems: The Next Computing Revolution," in *Proc. of the 47th Design Automation Conference*, New York, 2010.